**Eduardo Augusto Martins Rosa**

Licenciado em Engenharia Eletrotécnica e de Computadores

# Decomposição de Redes de Petri *IOPT*

Dissertação para obtenção do Grau de Mestre em

Engenharia Eletrotécnica e de Computadores

Júri:

Presidente: Doutor Rodolfo Alexandre Duarte Oliveira FCT/UNL

Arguente: Doutor Luís Filipe dos Santos Gomes, FCT/UNL

Vogal: Doutora Anikó Katalin Horváth da Costa, FCT/UNL

Orientador/a: Doutora Anikó Costa, Professora Auxiliar, Faculdade de Ciências e Tecnologias da Universidade Nova de Lisboa

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

**Setembro, 2018**

## Decomposição de Redes de Petri IOPT

# Agradecimentos

Concluindo esta dissertação é marcada a finalização de uma etapa importante na minha vida, pelo que gostaria de expressar a gratidão que sinto por todos os que contribuíram para isto, directa ou indirectamente.

À minha orientadora, Doutora Professora Anikó Katalin Horváth da Costa pela oportunidade, disponibilidade e toda a dedicação ao longo deste periodo.

Ao Professor Fernando Pereira (ISEL - Instituto Superior de Engenharia de Lisboa) pela disponibilidade e apoio durante a fase de implementação do trabalho desenvolvido.

Aos meus pais pelos valores transmitidos, pela dedicação e por todo o apoio incondicional pelo qual estarei eternamente grato. à minha irmã e avós por me terem acompanhado em todas as etapas da minha vida.

Aos meus amigos pela força que me deram e pelos momentos bem passados ao longo destes anos.

# Resumo

Com a crescente complexidade observada nos sistemas digitais, o recurso a formalismos de desenvolvimento de sistemas e às ferramentas que estes oferecem como suporte, torna-se a solução mais viável para um aumento da produtiviadde.

As redes de Petri, como formalismo de modelação de sistemas, possuem naturalmente a capacidade de descrever processos síncronos, paralelos, concorrentes, de partilha de recursos, mecanismos de composição e decomposição. Este formalismo possui uma representação gráfica, textual, e forte suporte matemático, o que permite o uso de técnicas e ferramentas de verificação, fundamentais na etapa de desenvolvimento.

Várias extensões às redes de Petri foram propostas. Estas extensões referem-se a adições de novas semânticas e funcionalidades que procuram oferecer suporte na descrição de determinadas características de um sistema em particular, tal como as redes de Petri *IOPT* (*Input-Output Place-Transition)* que procuram satisfazer os requisitos para suporte à modelação de sistemas digitais.

Considerando o modelo global de um sistema constituído por vários componentes descrito por uma única rede de Petri, procura-se com técnicas de decomposição, dividir esta rede em sub-redes (cada uma associada a um componente independente do sistema), de modo a facilitar tanto a análise dos componentes individualmente, como também, para preparar o controlador de cada componente para ser implementado numa plataforma específica.

Neste trabalho é apresentado um estudo sobre a semântica das redes de Petri *IOPT* , métodos de decomposição de redes de Petri e uma proposta de implementação do protótipo de uma ferramenta que automatiza o processo de decomposição da rede descritiva de um sistema global em várias sub-redes (componentes), com base no método operação *Net Splitting*.

**Palavras-Chave**: Redes de Petri IOPT, ferramenta de Decomposição de Redes de Petri IOPT, Sistemas Distribuídos, IOPT-Tools.

# Abstract

With the increasing complexity observed in systems, the use of model-based systems development, and the tools they provide as support, becomes a feasible solution to increase productivity.

Petri Nets, as a system formalism, naturally have the ability to describe synchronous, parallel, concurrent processes, resource sharing, composition and decomposition mechanisms, among others. This formalism has a graphical and textual representation and strong mathematical support, which allows the use of verification techniques and tools, necessary in the development stages.

Several extensions to Petri nets have been proposed. These extensions refer to additions of new semantics and functionalities that seek to support the description of certain characteristics of a particular system, just as the Petri Nets *IOPT (Input-Output-Place-Transition)* seek to satisfy the requirements to support the modelling of embedded and distributed automation systems.

Considering the global model of a system composed of several components described by a Petri net, it is sought with decomposition methods to divide a net into various sub-nets (each associated with a single component), in order to facilitate both the analysis of the components individually, as well as to prepare the controller of each component to be implemented on a specific platform.

This work presents a study on the semantics of *IOPT* Petri Nets and decomposition methods of Petri Nets. It's also proposed a prototype implementation of a tool that automates the decomposition process of a global system descriptive net, in several sub-nets (components), based on the *Net Splitting* operation method.

**Keywords:** *IOPT* Petri Nets, IOPT Petri Net Decomposition Tool, Distributed Systems, IOPT-Tools.

# Índice de Matérias

# Ìndice de Figuras

# Índice de Tabelas

# Lista de abreviaturas, siglas e símbolos

## Abreviaturas

*FPGA – Field-Programmable Gate Array*

*GALS – Globalmente Assíncrono Localmente Síncrono*

*IOPT – Input-Output Place-Transition*

*PNML – Petri Net Markup Language*

*UML – Unified Modeling Language*

*VHSIC – Very High Speed Integrated Circuits*

*VHDL – VHSIC Hardware Description Language*

*XML – eXtensible Markup Language*

## Siglas

FCT – Faculdade de Ciências e Tecnologias

UNL – Universidade Nova de Lisboa

# Capítulo 1 - Introdução

## 1.1 Motivação

Com a elevada complexidade presente em vários tipos de sistemas digitais (sistemas embutidos, sistemas de automação, entre outros), o uso de modelos, com base em formalismos para análise e especificação do comportamento de sistemas, teve um grande impacto positivo não só na produtividade do desenvolvimento, devido às ferramentas presentemente disponíveis para estes efeitos (como ferramentas de verificação e geradores de código), mas também na comunicação entre equipas envolvidas neste mesmo desenvolvimento.

Estes tipos de sistemas são constituídos por vários componentes que comunicam entre si síncrona ou assíncronamente. Cada componente é caracterizado como um dispositivo com vários pontos de acesso que representam a comunicação deste com o ambiente em que se insere. No entanto, torna-se contra-produtivo a modelação individual do controlador de cada componente sem ter presente em mente o sistema global ao qual esses componentes pertencem.

Numa primeira fase é necessário desenvolver o controlador do modelo global do sistema (controlo centralizado). Dentro do modelo do sistema global especificado é necessário identificar os diversos componentes concorrentes e aplicar-se uma decomposição de modo a obter-se os controladores individuais de cada sem qualquer alteração do comportamento inicial do sistema. Numa fase final é considerado o mapeamento de cada um dos controladores dos componentes numa plataforma específica.

Vários formalismos de especificação do comportamento de sistemas mostram-se adequados para a descrição deste tipo de sistemas, nomeadamente formalismos de especificação de controlo baseasdos em estados como, por exemplo, diagramas *UML (Unified Modeling Language)* [2] e redes de Petri. Os diagramas UML trata-se de um conjunto de tipos de diagramas, tais como diagramas de sequência, diagramas de estado, diagramas de actividade, entre outros e têm como objetivo oferecer suporte visual ao desenvolvimento de *Software*, modelação e documentação de sistemas. Estes diagramas têm a capacidade de descrever a estrutura, objectos e comportamentos dos sistemas.

Devido ao seu forte suporte matemático e semânticas bem definidas, o uso das redes de Petri traz enormes vantagens no desenvolvimento deste tipo de sistemas. Este suporte permite: a utilização de técnicas de verificação de modelos, para propósitos de validação das propriedades do modelo antes da etapa de prototipagem dos controladores físicos; a definição de transformações automáticas ou semi-automáticas para obtenção de novos modelos em diferentes níveis de abstracção; a aplicação de técnicas de decomposição de um modelo em sub-modelos, como para efeitos de análise ou preparação para implementação.

Várias classes de redes de Petri foram propostas de modo a expandir o número de características de sistemas que estas conseguem descrever. A classe *IOPT* permite a inclusão de sinais exteriores na sua estrutura que possibilitam a comunicação de um sistema descrito com o meio envolvente, tal como necessário na modelação e desenvolvimento de sistemas embutidos e de automação.

Vários métodos de decomposição de redes de Petri foram propostos com o propósito de facilitar a análise de cada componente de um sistema, focando-se na preservação de propriedades e aquisição de sub-modelos funcionais. No entanto, o método de interesse para implementação de uma ferramenta de decomposição automática, foca-se na capacidade do sistema decomposto resultante ser executado síncrona e distribuídamente com especificação sobre a comunicação entre os componentes constituíntes, mantendo o comportamento inicial, para uma futura eventual implementação física dos controladores.

A existência de ferramentas que permitem gerar código C e VHDL a partir de modelos de redes de Petri, pronto a ser implementado na plataforma alvo sem necessidade de código adicional de baixo nível, simplifica a fase imediatamente anterior à prototipagem do controlador físico de cada componente.

## 1.2 Objetivos

Os principais objetivos deste trabalho são realizar um estudo sobre a semântica das redes de Petri *IOPT* e implementar um protótipo de uma ferramenta capaz de efectuar uma decomposição automática de uma rede de Petri *IOPT* em várias sub-redes associadas aos controladores dos componentes isolados do sistema.

A ferramenta implementada tem como base o método proposto em [1], que por sua vez toma a classe de redes de Petri *IOPT* como referência e visa oferecer suporte no desenvolvimento de

sistemas embutidos. Este método procura, além da decomposição, um meio de comunicação entre os componentes(por meio de canais síncronos direccionais, permitindo uma execução síncrona dos componentes gerados. Contudo, pretende-se automatizar o processo de decomposição, reduzindo o tempo e aumentando a produtividade no que toca ao desenvolvimento dos controladores de um modelo decomposto a partir de um modelo global. Também pretende preparar individualmente cada componente gerado para ser implementado num dispositivo alvo e possibilitar o uso do resultado da decomposição como preparação para uma eventual posterior adopção de uma arquitetura Globalmente Assíncrona e Localmente Síncrona (GALS).

## 1.3 Organização e Estrutura da Dissertação

Este documento encontra-se dividido em seis Capítulos. No primeiro Capítulo é dada uma breve introdução sobre o que foi realizado ao longo deste trabalho bem como os principais objetivos do mesmo.

No segundo Capítulo é realizado um estudo sobre redes de Petri. São abordadas várias definições comuns a todas as classes de redes de Petri e mencionados vários tipos de classes, embora o estudo se foque principalmente na classe *IOPT*. No final do mesmo capítulo é também apresentada uma introdução à plataforma usada para modelação de controladores de sistemas, utilizando a classe *IOPT* como referência. Esta plataforma é alvo de uma integração com a ferramenta de decomposição de redes (SPLIT) implementada neste trabalho. A plataforma em questão foi utilizada também para gerar todos os exemplos de modelos de redes de Petri representados nas Figuras no decorrer deste documento.

No terceiro Capítulo é dada uma introdução a métodos de Decomposição de redes de Petri investigados pelo autor e é estudado em maior detalhe o método Operação *Net Splitting*, no qual se baseia a implementação da ferramenta de decomposição automática.

No quarto Capítulo são apresentadas todas as ferramentas, recursos e Algoritmos utilizados na implementação da ferramenta SPLIT. Também é apresentado um conjunto de instruções de utilização da ferramenta no ambiente da plataforma de desenvolvimento onde se encontra integrada.

No quinto Capítulo é feita uma apresentação dos resultados da aplicação da ferramenta implementada em dois exemplos de controladores de sistemas distintos. Ao longo deste capítulo

é feita uma descrição detalhada do comportamento dos exemplos mencionados. Os resultados esperados são posteriormente comparados com o resultado obtido de modo a validar a utilização da ferramenta *SPLIT*.

No sexto e último Capítulo são expostas as principais conclusões e sugestões para trabalhos futuros.

# Capítulo 2 - Redes de Petri

## 2.1 Introdução

As Redes de Petri são uma ferramenta/linguagem de especificação/modelação gráfica e matemática do comportamento de sistemas introduzida por Carl Adam Petri documentadas na sua tese de doutoramente em 1962. Este formalismo foi primeiramente utilizado na descrição de processos químicos.

Do ponto de vista gráfico, as redes de Petri possuem um suporte algo semelhante a gráficos de fluxo, diagramas de blocos ou de estados [3]. Estas características permitem uma fácil descrição visual e entendimento do comportamento de um sistema. Marcações (*Tokens*) são utilizadas para simulação das actividades dinâmicas e concorrentes do sistema em causa, para propósitos de validação.

As redes de Petri possuem também uma representação textual normalizada sobre o formato *PNML (Petri Net Markup Language*), um forte suporte matemático e semânticas bem definidas, sendo que é possível a sua representação por meio de equações algébricas e sob a forma de outros formalismos matemáticos. Devido a este forte suporte matemático é possível usar métodos formais para análise e verificação das propriedades da rede, quer sejam comportamentais ou estruturais. Estas características trazem enormes vantagens do ponto de vista de implementação de ferramentas, sendo este um dos objetivos deste trabalho.

Novas definições de Redes de Petri têm vindo a ser propostas. Estas tomam como base o conceito de redes de Petri inicial, adicionando novas extensões e características de rede que possam satisfazer o propósito que essa rede pretende definir. Neste capítulo, serão abordadas várias definições básicas formais acompanhadas de representações gráficas dos diversos exemplos construídos, características das redes, e embora mencionadas, apenas se terá especial foco o estudo da semântica das classes de redes de Petr as quais são de especial interesse neste trabalho, como é o caso das redes de Petri *IOPT*.

## 2.2 Redes de Petri: Definições Básicas

No seu aspecto mais básico e como apresentado da definição 1 a 3, que se encontram no decorrer deste capítulo, as redes de Petri são caracterizadas como um diagrama bipartido dirigido [3]. Estas são constituídas por dois tipos de nós distintos. Estes nós encontram-se interligados por meio de arcos, sendo que não existe ligação entre nós do mesmo tipo. Estes nós são definidos como lugares (elemento passivo) e transições (elemento activo) e encontram-se associados directamente às características estáticas e dinâmicas do sistema, respetivamente. As definições apresentadas foram escolhidas de [4], sendo as mesmas comuns a todas as classes de redes de Petri:

**Definição 1**. *(de [4]) Seja P e T dois conjuntos disjuntos, e seja* $F \subseteq (P \times T) \cup (T \times P)$. *Então* $N = (P, T, F)$ *é uma rede.*

*P* representa os lugares (Elemento passivo), *T* representa as transições (elemento activo) e F representa os arcos que por sua vez representam relações entre nós de tipos distintos e indicam a direcção do fluxo da informação.

**Notação 1.** (*de [4] adaptada por [1]) P, T, F são respetivamente denotados como N.P, N.T, N.F. Para se excluir possíveis confusões, N representa o conjunto* $P \cup T$ , *e* $aFb$ *o conjunto* $(a,b) \in F$ .

**Definição 2.** *(de [4] adaptado por [1]) Seja* $N = (P, T, F)$.

- $F^{-}, F^{+}$ *e* $F^{*}$ *denotam a relação inversa, o fecho transitivo e o fecho reflectivo e transitivo respetivamente:*
  - $aF^{-1}b$ *se* $bFa$*;*
  - $aF^{+}b$ *se* $aFc_1Fc_2 \ldots c_nFb$ *para* $c_1, c_2, \ldots c_n \in N$*;*
  - $aF^{*}b$ *se* $aF^{+}b$ *sendo* $a = b$*;*

*Para* $a \in N$, *seja* $F(a) = \{b \mid aFb\}$.

- Quando F pode ser assumido a partir do contexto, para $a \in N$ escreve-se $\bullet a$ em vez de $F^{-1}(a)$ e $a \bullet$ em vez de $F(a)$. Esta notação é traduzida em sub-conjuntos $A \subseteq N$ por $\bullet A = \bigcup_{a \in A} \bullet a$ e $\bullet A = \bigcup_{a \in A} a \bullet$:
  - $\bullet A$ é o pré-conjunto ou conjunto de elementos de entrada de A;
  - $A \bullet$ é o pós-conjunto ou conjunto de elementos de saída de A;

**Definição 3.** *(de [4] adaptado por [1]) Seja N uma rede.*

- *Para $x \in N$, N diz-se isolada se $\bullet x \cup x \bullet = \emptyset$;*
- *N diz-se conexa se para todos os $x, y \in N : x(F \cup F^{-1})^* y$;*
- *N diz-se fortemente conexa se para todos os $x, y \in N : x(F^*)y$;*

Como é possível observar na *Figura 2.1*, que ilustra os vários tipos de redes referidos na **definição 3**, a representação visual usada para os lugares é normalmente feita por meio de círculos, das transições por rectângulo ou quadrados e os arcos por meio de setas negras.

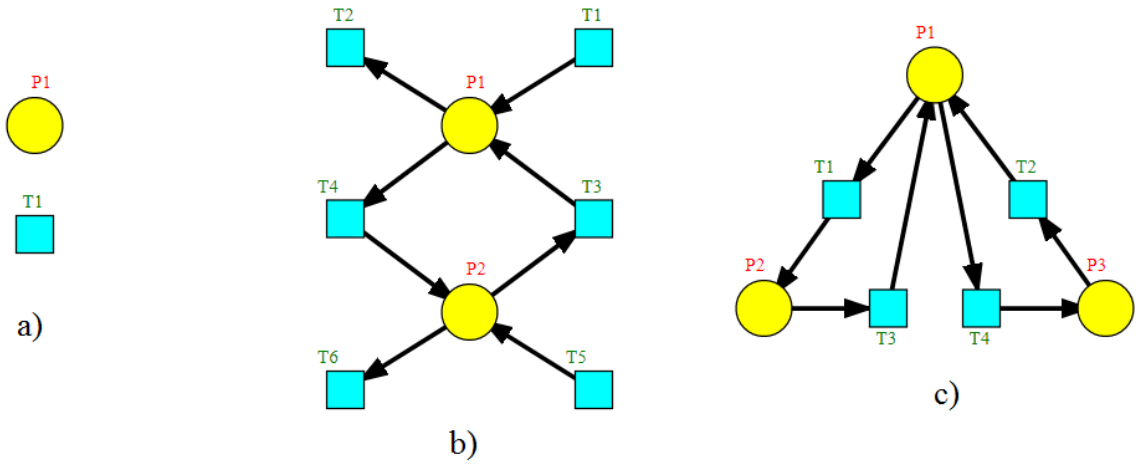


Figura 2.1 - Tipos de Redes; a) Rede isolada; b) Rede conexa; c) Rede fortemente conexa;

A um lugar que possua um arco direccionado para uma transição, denominamos um lugar de entrada dessa mesma transição. A um lugar que tenha a si ligado um arco com origem numa transição denominamos como lugar de saída dessa transição.

Para além da rede acima descrita, uma rede de Petri também inclui na sua estrutura uma marcação inicial (estado inicial do sistema), que se dá por meio de distribuição de marcas pelos lugares da rede. As *marcas* são normalmente representadas por pontos negros ou pelo valor inteiro correspondente ao número de marcas atribuídas a esse lugar, como se pode observar na *Figura 2.2*. A distribuição destas marcas pelos lugares da rede define o estado em que o sistema (representado pela rede) se encontra.

Nas redes de Petri Condição-Evento [5], uma transição só se encontra habilitada para disparar quando todos os lugares que a precedem (lugares de entrada) possuírem uma marca. Quando todos os seus lugares de entrada tiverem preenchidos com uma marca, essa transição pode ou não disparar. Quando uma transição dispara remove a marca dos seus lugares de entrada e coloca um uma marca em todos os seus lugares de saída. Neste tipo de redes de Petri os lugares são vistos como condições e as transições como eventos.

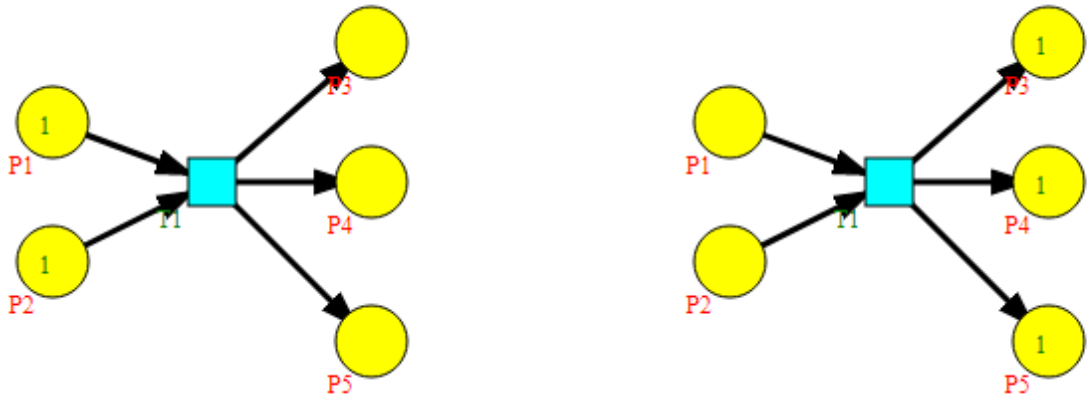


Figura 2.2 - Resultado do disparo de uma transição numa rede de Petri Condição-Evento.

Esta regra de disparo pode diferir ligeiramente dependendo da classe de rede de Petri em causa. Nas redes de Petri Condição-Evento, como já referido, apenas se considera o peso (ou dimensão) dos arcos como 1, em que cada lugar só pode possuir zero ou uma marca no máximo. No caso da classe Place - Transition [5] (generalização das anteriores) os arcos podem ter associados pesos superiores a um e os lugares podem possuir zero, uma ou mais marcas. Para uma transição estar

habilitada para disparar é necessário que os seus lugares de entrada possuam um número de marcas igual ao peso do arco que liga esse lugar de entrada à transição.

Quando a transição dispara é removido um número de marcas correspondente ao peso do arco de ligação entre o lugar de entrada em causa, e é colocado nos lugares de saída um número de marcas correspondente ao peso do arco que liga a transição ao lugar de saída, como exemplificado na *Figura 2.3*.

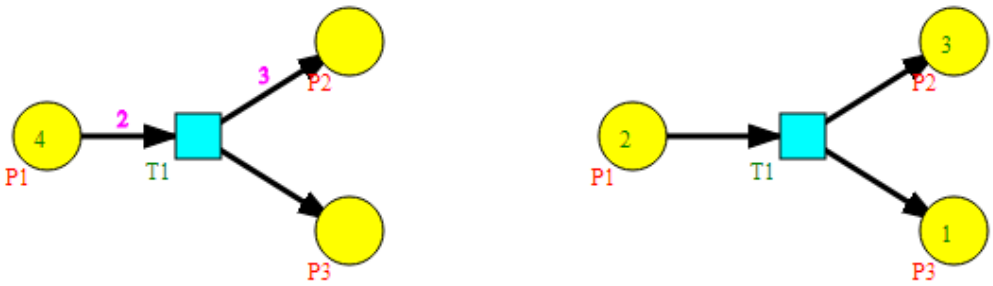


Figura 2.3 - Resultado do disparo de uma transição numa rede de Petri Place-Transition.

**Definição 4 (Rede de Petri *Place-Transition*).** *(adaptado de [3]) Uma rede de Petri Place-Transition (PT) é definida por um tuplo N = (P, T, F, W ), onde:*

*- P, é um conjunto finito de lugares;*

*-T, é um conjunto finito de transições;*

*-* $F \subseteq (P \times T) \cup (T \times P)$ *, é um conjunto de arcos (Relação de fluxo);*

*-W, é a função de peso,* $W: F \rightarrow N_0$*;*

$-P \cup T = \emptyset \ e \ P \cap T = \emptyset$ *;*

As redes de Petri têm a capacidade de descrever processos concorrentes, distribuídos, paralelos, sequenciais, entre outros[3]. No entanto, existem situações que impõe um problema na evolução de uma rede. Estas situações que podem ser observadas na *Figura 2.4. a)*, são chamadas de conflito e ocorrem quando todas as transições de saída de um lugar encontram-se habilitadas para

disparar, mas o disparo de uma das transições impossibilita o disparo das outras transições dentro da mesma situação de conflito.

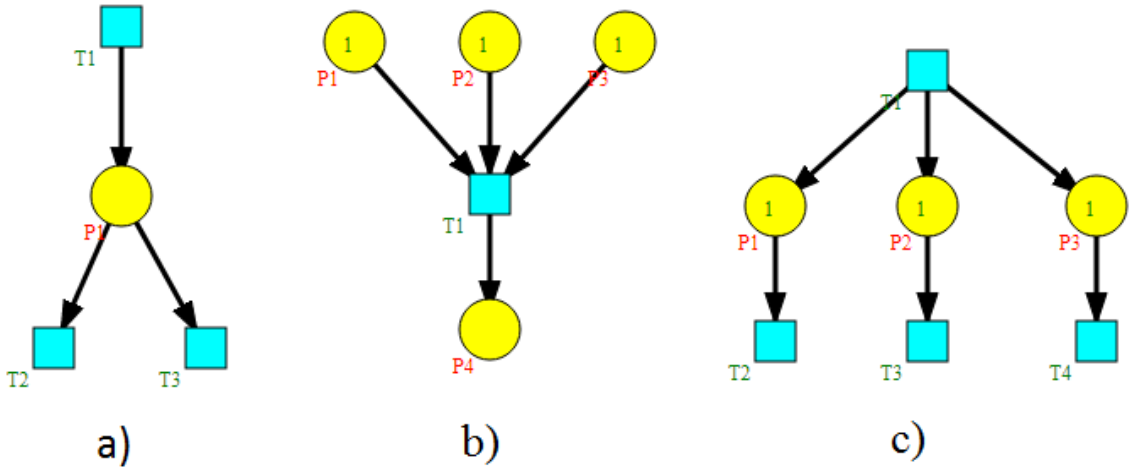


Figura 2.4 - a) Conflito; b) Sincronização; c) Concorrência.

Na Figura 2.4 b) e c) encontram-se representadas descrições de processos de sincronização e concorrentes respetivamente.

## 2.3 Tipos de Classes de Redes de Petri

Como observado na transição da classe Condição-Evento para a *Place-Transition*, as redes de Petri podem ser suplementadas com algumas funcionalidades adicionais, de modo a satisfazer um propósito específico com a introdução de novas semânticas, definições, mecânicas que tornem a descrição do comportamento de um sistema, em particular possível e mais simples.

Várias Classes de Redes de Petri têm vindo a ser propostas, e em [6] estas classes são divididas em dois tipos distintos. Os tipos distintos identificados referem-se às dependências do comportamento da rede de Petri com o ambiente que a envolve e a outras características que serão mencionadas.

### 2.3.1 Redes Autónomas

As Redes Autónomas não possuem qualquer dependência por parte de condições ou eventos

externos do ambiente em que se inserem, sendo que, a sua execução depende apenas da sua marcação (estado) da rede. As Classes de Redes de Petri deste tipo podem ainda ser divididas em três níveis. Na ***Tabela. 1,*** é mostrado os três níveis das classes deste tipo juntamento com uma descrição e exemplos.

Tabela 1 - Tipos de redes de Petri autónomas**.**

| Redes de Baixo Nível | | |
|---|---|---|
| Nível | Descrição | Exemplos |
| 1º Nível | - Classes em que os lugares podem possuir zero ou uma marca no máximo, sem qualquer informação associada;<br>- Lugares representam condições;<br>- Transições representam eventos; | - Redes de Petri Condição-evento [5].<br>- Redes de Petri Elementares [7].<br>- Redes Petri de Livre Escolha [8]. |
| 2º Nível | - Classes em que os lugares podem possuir um número superior a um de marcas. Os arcos, também podem ter a si associados um peso superior a um;<br>- Lugares representam contadores;<br>- Normalmente usadas na especificação de sistemas de controlo, automação e comunicação; | - Place-Transition Petri Nets[5]. |
| Redes de Alto Nível | | |
| 3º Nível | - Cada marca pode possuir informação, estruturas associadas diferenciando-o de todas as outras marcas. | - Redes de Petri Coloridas[9];<br>- Redes de Petri com Marcas Individuais[10];<br>- Redes de Petri Objecto[11];<br>- Redes de Petri Predicado/Transição [12]; |

### 2.3.1 Redes Não-Autónomas

Com a necessidade de adequar as redes para modelação de aspectos importantes e específicos (que as Autónomas não têm capacidade de) tais como a descrição de dependências que um sistema possa ter com o ambiente em que se insere, dependências temporais, entre outras características.

Várias Extensões foram propostas. Em [6] estas extensões são divididas em três tipos, cada com um objetivo e características diferentes, tais como:

1) Capacidade de integrar no grafo referências a características do sistema em que a rede se insere tais como sinais de controlo, como no caso das redes de Petri sincronizadas [13] e as redes de Petri Interpretadas[14];

2) Capacidade de testar qualquer estado de marcação, como por exemplo, as classes de arco inibidor e prioridade [15] , em que a associação de prioridades relativas a transições oferece uma solução automática para eventuais conflitos que a rede possa possuir;

3) Capacidade de integrar dependências temporais tal como nos casos das redes de Petri Estocásticas Generalizadas [16] e Redes Coloridas Temporizadas [9] que tomam como referência as redes de Petri Coloridas (Rede Autónoma).

No entanto, as classes mencionadas procuram arranjar uma solução para um problema em específico, pois incorporam em si apenas uma das característica acima mencionadas, não sendo suficiente para se modelar o comportamento de determinados tipos de sistemas tais como, por exemplo, sistemas embutidos. Com este tipo de sistemas em mente, foi proposto em [17], a classe redes de Petri *IOPT* (Input-Output-Place-Transition). Esta classe procura incorporar todas as características necessárias à sua modelação e será alvo do foco deste estudo.

## 2.4 Redes de Petri *IOPT (Input - Output Place - Transition)*

### 2.4.1 Introdução

Considerando o controlador de um componente cuja execução se encontra dependente de restrições impostas pelo ambiente em que que se insere, existe a necessidade de se recorrer a extensões, tais como, a capacidade da rede que representa o controlador de determinado componente especificar comunicação com o meio via sinais e eventos de entrada e de saída. As redes de Petri *IOPT* surgem com o propósito de oferecer suporte ao desenvolvimento de sistemas embutidos, robóticos e de automação. Os sistemas embutidos referidos são caracterizados, em

[18], como sistemas de eventos discretos que acomodam restrições em tempo-real e capacidade de processamento de dados.

A classe *IOPT* toma como base a definição da classe *Place - Transition* de redes de Petri que são normalmente usadas na modelação de controlo de automação e extendem-nas, integrando nestas a capacidade de interagirem com o ambiente recorrendo a sinais e eventos de entrada e de saída [18].

Mais detalhadamente, os sinais de entrada podem ser usados para se obter informação do ambiente, como por exemplo, leitura de sinais provenientes de outros sistemas, leitura do estado de sensores, entre outros[19]. Os eventos de entrada e de saída encontram-se, por sua vez, associados a alterações nos valores dos sinais. Estas alterações nos sinais de entrada provocam o disparo de eventos de entrada e o disparo de eventos de saída causam alteração dos valores de sinais de saída. Os sinais de saída podem ser usados para enviar informação para outros sistemas, para manipulação de actuadores, iluminação de LEDS, etc...

Numa versão mais recente da classe redes de Petri *IOPT*, apresentada em [17], além da interacção com o sistema controlado, inclui outras características adicionais à definição das redes de Petri Place-Transition. Sendo estas:

- *Prioridades em transições* - permite uma solução rápida em situações de conflito entre transições;
- *Atributo fixo para lugares* - com o propósito de fornecer informação relevante de implementação a uma ferramenta que gere código automático;
- *Especificação para conjuntos de transições em situação conflito* - engloba a especificação de conjuntos de transições em situação de conflito, tal como o nome indica;
- *Especificação para conjuntos de transições síncronos* – engloba especificação para conjuntos de transições síncronas, ou seja, com semânticas de fusão segundo [1], que significa que as transições dentro de um conjunto comportam-se como se fossem só uma;
- *Arcos de teste* - permite uma arbitragem justa em situações de conflito entre transições;

O exemplo da *Figura 2.5* ilustra uma Rede de Petri *IOPT* que modela o controlador de um sistema constituído por duas vagonetes que se deslocam entre dois pontos (ponto de carga, e o ponto de descarga). No sistema mencionado, as características desejadas são as seguintes:

- Inicialmente as vagonetes encontram-se paradas no ponto de carga;
- Quando é dada a ordem de deslocamento, todas as vagonetes movem-se até ao ponto de descarga;

- Quando é dada nova ordem de deslocamento, todas as vagonetes movem-se até ao ponto de carga;
- Tanto no deslocamento para o ponto de descarga como para o ponto de carga, a reacção à ordem de deslocamento é efectiva quando todas as vagonetes tiverem chegado ao ponto presente respetivo (destino da última ordem de deslocamento).

Na seguinte rede, os sinais de entrada *GO, BACK, A1, A2, B1, B2* são representados na legenda por meio de cícrulos azuis. Quando ambas as vagonetes se encontram no ponto descarga, ou seja, existe uma marcação nos lugares *car1_ready* e *car2_ready*, e a guarda do sinal de entrada *GO* apresenta valor verdadeiro (*GO* = 1), a transição encontra-se habilitada para disparar. Com o disparo da transição *GO*, ambos lugares *car1_move_forward* e *car2_move_forward* são marcados. A marcação nestes lugares representa o estado do sistema em que as vagonete se deslocam até ao ponto de carga. A estes lugares encontram-se atribuídas acções associadas a sinais de saída. Os sinais de saída *M1, M2, Dir1 e Dir2* são representados na legenda por círculos verdes que se encontram associados a lugares. *M1 e M2* representam os sinais de saída gerados após ordem de deslocamento, ou seja, quando *M1* e *M2* são verdadeiros, as vagonetes movimentam-se. *Dir1 e Dir2*, representam a direcção do movimento, se *Dir1 e Dir2* apresentam valor verdadeiro (Dir1 = 1, Dir2 = 1), então o sentido do movimento será do ponto de descarga para o ponto de carga. Caso apresentem valor falso a direcção será a inversa. Quando as vagonetes alcançam o ponto de carga, as guardas dos sinais B1 e B2 apresentam valor verdadeiro (B1 = 1, B2 = 1). Quando disparadas as transições B1 e B2, os lugares car1_arrived e car2_arrived são respetivamente marcadas. Estes lugares representam o estado do sistema em que as vagonete se encontram paradas no ponto de carga. Quando ambas as vagonetes se encontram no ponto de carga e a guarda do sinal *BACK* apresenta valor verdadeiro, os lugares *car1_move_back* e *car2_move_back* são marcados. A marcação nestes lugares representa o movimento das vagonetes do ponto de carga de volta ao ponto de descarga. Quando as vagonetes alcançam o ponto de carga, a guarda dos sinais A1 e A2 apresentam valor verdadeiro. Quando as transições A1 e A2 disparam, os lugares *car1_ready* e *car2_ready* são marcados (voltando ao estado inicial).

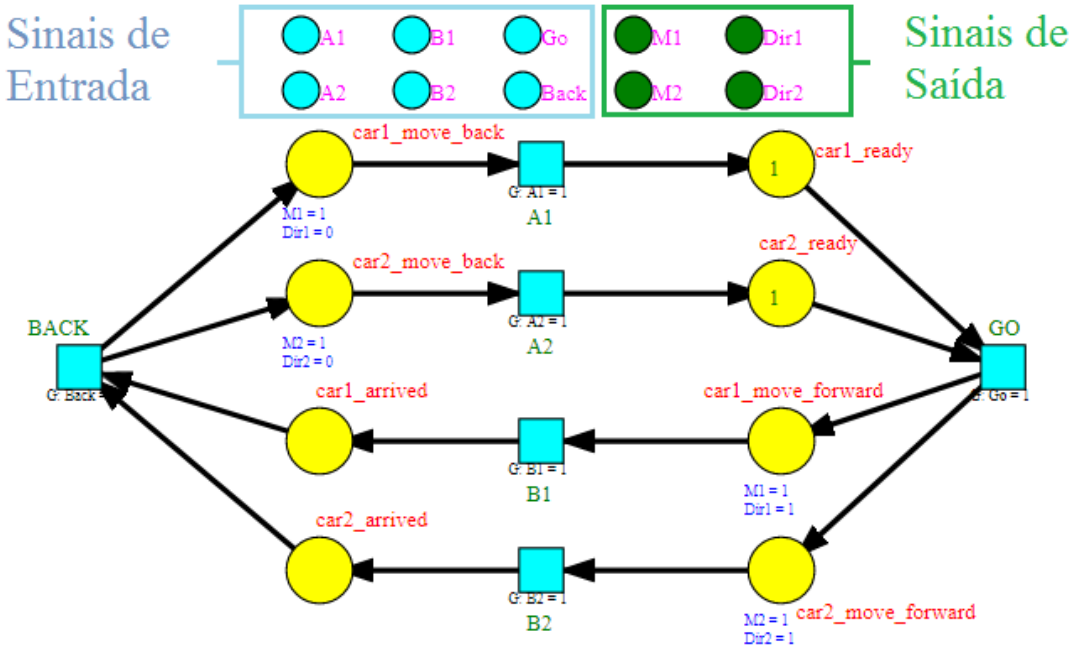


Figura 2.5 - Rede *IOPT* de um sistema de duas vagonetes.

## 2.4.2 Interação do Controlador com o Meio

Os controladores dos componentes de um sistema podem ser caracterizados, cada, em dois componentes principais [18], sendo o primeiro relativo à descrição da interacção física com o controlador (interface) e o segundo ao modelo comportamental caracterizado pela rede *IOPT*.
O interface do controlador traduz-se, como já discutido anteriormente, num conjunto de sinais activos de saída e de entrada, e de eventos:

**Definição 5.** *(de [17) O Interface de um sistema com uma rede IOPT é um tuplo* $ICS = (IS, IE, OS, OE)$ *que satisfaz os seguintes requerimentos:*

1. *IS é um conjunto finito de sinais de entrada;*
2. *IE é um conjunto finito de eventos de entrada;*
3. *OS é um conjunto finito de sinais de saída;*
4. *OE é um conjunto finito de eventos de saída;*

*5.* $IS \cap IE \cap OS \cap OE = \emptyset$*;*

**Definição 6.** *(de [18]) Dado um interface* $ICS = (IS, IE, OS, OE)$ *com um sistema , o estado de entrada de um sistema é definido por um par* $SIS = (ISB, IEB)$ *que satisfaz os seguintes requerimentos:*

*1. ISB é um conjunto finito de ligações de sinais de entrada:* $ISB \subseteq IS \times N_0$*;*

2. IEB é um conjunto finito de ligações de eventos de entrada: $IEB \subseteq IE \times B$;

### 2.4.3 Definições da Classe *IOPT*

A definição seguinte referente à classe redes de Petri *IOPT* assume o uso de uma linguagem de inscrição como uma sintaxe concreta para permitir a especificação de expressões algébricas, variáveis e funções que por sua vez permitem uma descrição das guardas associadas a transições e condições de acções de saída associadas a lugares [17].
O conjunto de expressões Booleanas é representado por *BE* e a função *Var(E)* retorna o conjunto de variáveis de uma determinada expressão *E [17]*.

**Definição 7 (Rede de Petri *IOPT*)**. (de [17]) *Dado um controlador com interface* $ICS = (IS, IE, OS, OE)$ *, uma rede IOPT é um tuplo* $N = (P, T, A, TA, M, weight, weightTest, priority, isg, ie, oe, osc)$ *que satisfaz os seguintes requerimentos:*

1. ***P*** *é um conjunto finito de lugares;*
2. ***T*** *é um conjunto finito de transições;*
3. ***A*** *é um conjunto de arcos, tal que* $A \subseteq ((P \cup T) \times (T \cup P))$ *;*
4. ***TA****, é um conjunto de arcos de teste, tal que* $TA \subseteq (P \cup T)$ *;*
5. ***M****, representa a função de marcação:* $M: P \rightarrow N_0$*;*
6. ***weight****, representa o peso dos arcos:* $A \rightarrow N_0$*;*
7. ***weighTest****, representa o peso de arcos de teste:* $TA \rightarrow N_0$*;*
8. ***priority****, representa prioridade de disparo e é uma função parcial que associa transições a inteiros não negativos:* $priority: T \rightarrow N_0$*;*

9. ***isg*** *é uma guarda (função parcial) de um sinal de entrada que aplica transições a expressões booleanas (onde todas as variáveis são sinais de entrada):* $isg : T \to BE$*, onde* $\forall eb \in isg(T), Var(eb) \subseteq IS$ *;*

10. ***ie****, é uma função parcial de evento de entrada que aplica transições a eventos de entrada:* $ee : T \to IE$ *;*

11. ***oe****, é uma função parcial de evento de saída que aplica transições a eventos de saída: ee:* $T \to OE$

12. ***osc****, é uma função de condição de sinal de saída associada a lugares para um conjunto de regras (RULES): osc:.* $P \to \mathrm{P}(RULES)$ *, onde* $RULES \subseteq (BES \times OS \times N_0)$ *,* $BES \subseteq BE$ *e* $\forall e \in BES$ *,* $Var(e) \subseteq ML$ *com ML sendo o conjunto de identificadores para cada marcação de lugar após um dado passo de execução: Cada marcação de lugares tem um identificador associado, que é usado quando o código gerado é executado;*

Prioridades, eventos e sinais encontram-se associadas a transições. Estas também podem ter a si atribuídas funções de sinais de entrada (guardas).

As redes *IOPT* usam semânticas de passo máximo o que permite obter operações determinísticas e coerentes. Uma transição dispara apenas quando esta estiver habilitada, ou seja, todos os seus lugares de entrada possuem um número de marcas igual ou superior ao peso expresso dos arcos correspondentes, e quando os eventos de entrada e as guardas de sinais de entrada externos a si associados estiverem activos. Os sinais de saída podem ser alterados a partir de eventos de saída ou no final de cada passo de execução. A evolução da rede, de acordo com o paradigma de Sincronismo, só é possível em determinados instantes de tempo. A estes instantes de tempo denominamos tics que são definidos por um *Clock/relógio* global, e um passo de execução refere-se ao tempo entre dois tics.

Num passo de execução em que várias transições em situação de conflito estão preparadas para disparar, apenas a transição com maior prioridade associada irá disparar.

***Notação 2****.(adaptado de [1]) M(p) denota a marcação de um lugar p de uma rede com marcação M e* $\bullet t$ *denota os lugares de entrada de uma dada transição t ou de um dado conjunto de transições S:*

$\bullet t = \{p \mid (p,t) \in A\}$;
$\bullet S = \{p \mid (p,t) \in A \wedge t \in S\}$;
$\lozenge t = \{p \mid (p,t) \in TA\}$;
$\lozenge S = \{p \mid (p,t) \in TA \wedge t \in S\}$

**Definição 8 (Condição de Habilitação).** *(de [17]) Dada uma rede N = (P, T, A, TA, weight, weightTest, priority, isg, ie, oe, osc) e um sistema de interface ICS = (IS, IE, OS, OE) entre a rede N e um sistema com estados de entrada SIS = (ISB, IEB), uma transição t, sem qualquer conflito estrutural, está habilitada para disparar num determinado tic se as seguintes condições forem satisfeitas:*

1. $\forall p \in \bullet t, M(p) \geq weight(p,t)$;
2. $\forall p \in \lozenge t, M(p) \geq weightTest(p,t)$;
3. *A guarda do sinal de entrada da transição t deve estar avaliada como verdadeira (True) para uma dada ligação de sinal de entrada:* $isg(t) < ISB >$.
4. $(ie(t), true) \in IEB$;

*Uma transição que se encontre num conflito estrutural com outras transições, encontra-se habilitada para disparar apenas se esta tiver a maior prioridade dentro do respetivo conjunto de transições em situação de conflito CS:* $\forall t' \in CS, t' \neq t \Rightarrow priority(t') \leq priority(t)$.

**Definição 9 (Passo numa Rede *IOPT*).** *(de [18] adaptado por [1 ]) Dada uma rede N = (P, T, A, TA, weight, weightTest, priority, isg, ie, oe, osc) e um sistema de interface ICS = (IS, IE, OS, OE) entre a rede N e um sistema com estados de entrada SIS = (ISB, IEB). Seja* $ET \subseteq T$ *o conjunto de todas as transições habilitadas. Então, Y é um passo numa rede N se a seguinte condição for satisfeita:*

$$Y \subseteq ET \wedge \forall t_1 \in (ET \setminus Y), \exists SY \subseteq Y, (\bullet t_1 \bigcap \bullet SY) = \phi \wedge \exists p \in (\bullet t_1 \bigcap \bullet SY),$$
$$(weight(p,t_1) + \sum_{t \in SY} weight(p,t) > M(p))$$

**Definição 10 (Ocorrência de Passo e Marcação Sucessora).** *(de [17]) Dada uma rede N = (P, T, A, TA, weight, weightTest, priority, isg, ie, oe, osc) e um sistema de interface*

*ICS = (IS, IE, OS, OE) entre N e um sistema com estado de entrada SIS = (ISB, IEB), a ocorrência de um passo Y na rede N retorna a rede*

*N'=(P, T, A, TA, M', weight, weightTest, priority, isg, ie, oe, osc), igual à rede N com a excepção da marcação sucessora , que é dada pela expressão:*

$$M' = \{(p, m - \sum_{t \in Y \wedge (p,t) \in A} weight(p,t) + \sum_{t \in Y \wedge (t,p) \in A} weight(t,p)) \in (P \times N_0) \mid (p,m) \in M\}$$

## 2.5 *IOPT-Tools*

A ***IOPT-Tools*** [20], [21] é uma plataforma de desenvolvimento integrado de utilização gratuíta com um interface gráfico *web*. Esta plataforma possui um conjunto de ferramentas que suportam o desenvolvimento, teste e implementação de controladores de sistemas embutidos, sistemas de automação, entre outros sistemas. Esta plataforma de desenvolvimento usa como base a classe *IOPT* de redes de Petri, permitindo a comunicação da rede desenvolvida com um ambiente externo por meio de sinais de saída e entrada.

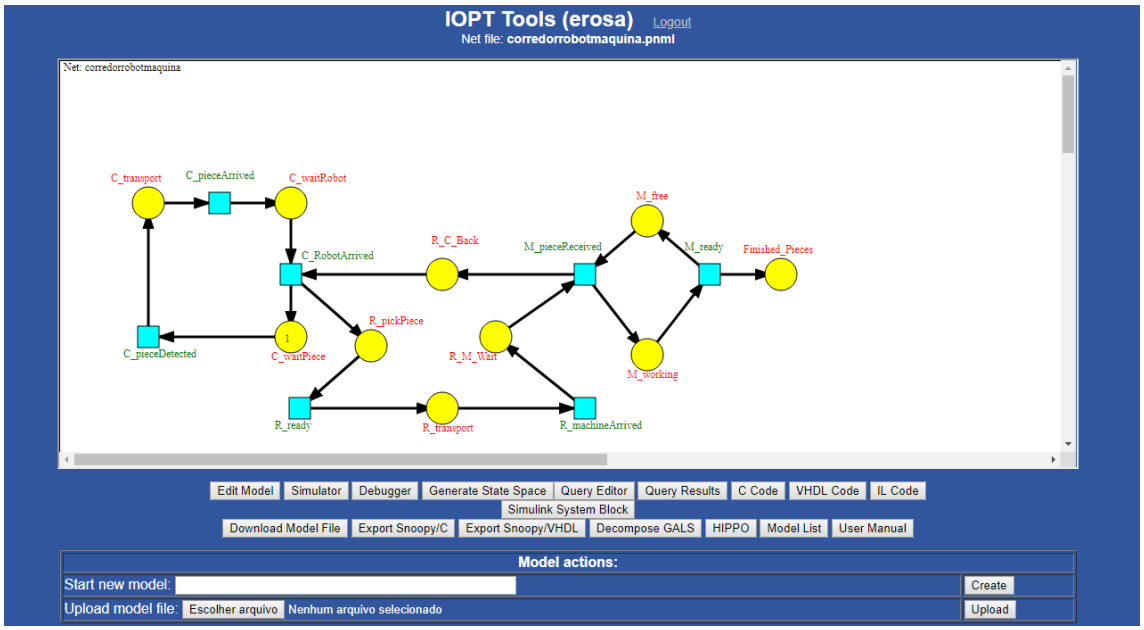


Figura 2.6 – Ambiente *Web* da plataforma *IOPT*-Tools.

Todas as ferramentas interactivas na estrutura desta plataforma são executadas usando princípios AJAX. No entanto, tanto o armazemento de ficheiros como o processamento de operações são efectuados no servidor.

Esta plataforma de desenvolvimento consiste em[19]:

- Editor Interactivo Gráfico - Para edição de modelos *IOPT* e responsável pela leitura, escrita e criação dos ficheiros PNML que são usados pela maioria das ferramentas da plataforma [22];
- Simulador - Para teste do comportamento dos modelos desenvolvidos;
- Ferramentas de verificação - Para automação do processo de verificação do modelo e de propriedades, permitindo a detecção e correcção de erros no design, antes da fase de implementação do protótipo [23];
- Geradores de código C, VHDL - Gera código de baixo nível, usado para ser implementado o controlador no dispositivo físico alvo, eliminando a necessidade de escrita manual de código adicional, com a excepção de certas operações como atribuição de pins em micro-controladores ou *FPGA*;

Dentro das ferramentas de verificação referidas que as *IOPT-Tools* oferece, estão incluídas um gerador e visualizador do espaço de estados e um sistema de consulta.

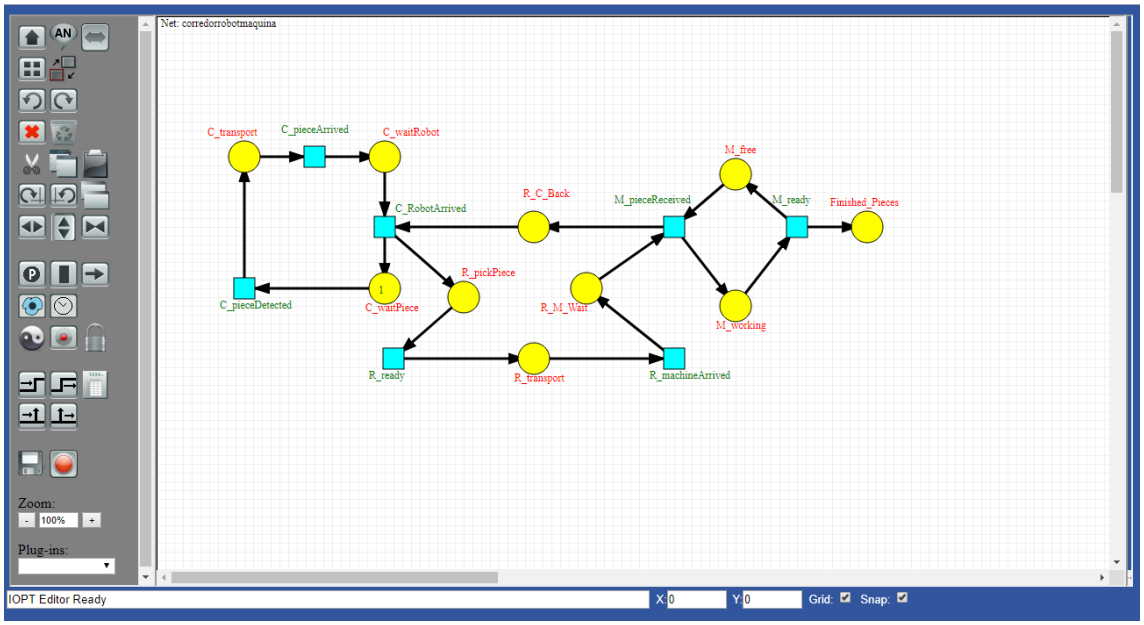


Figura 2.7 – Editor PNML da plataforma *IOPT*-Tools.

Além das características mencionadas, as *IOPT*-Tools incluem uma extensão das Redes de Petri *IOPT* que suporta o desenvolvimento de sistemas *GALS* (Globalmente Assíncrono Localmente Síncrono) com base no método proposto em [24]. Este método introduz nas redes *IOPT*, os conceitos de domínios temporais e de canais Assíncronos. Ambos os domínios temporais e canais assíncronos estão incluídos nas ferramentas de edição de modelos das *IOPT-Tools*. Os Canais assíncrono são representados, no editor, por nuvens azuis e os canais síncronos por círculos negros (Figura 2.8).

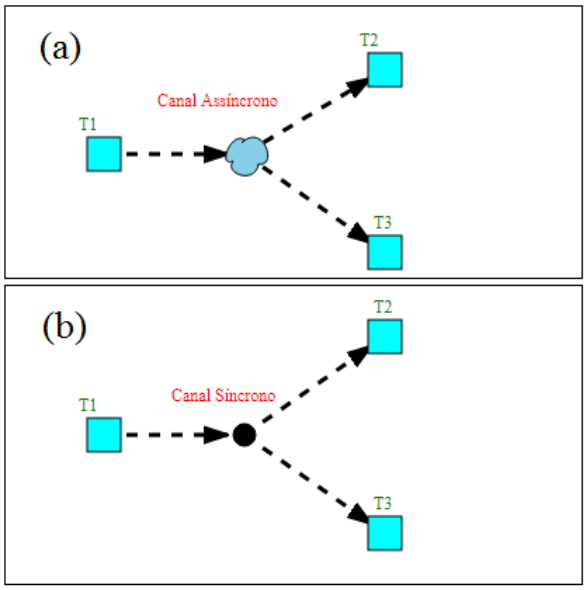


Figura 2.8 – Representação gráfica dos canais assíncronos (a) e síncronos (b) no editor da plataforma *IOPT*-Tools.

Os sistemas *GALS* são caracterizados como sistemas compostos por vários componentes síncronos que comunicam entre si assíncronamente. Recorrendo-se a um Clock Global na implementação do paradigma Síncrono para representar a comunicação entre componentes, deparamo-nos com um problema relativo à incompatibilidade deste paradigma na implementação de determinados tipos de sistemas. No entanto, devido ao elevado *overhead* e à falta de suporte ao desenvolvimento de soluções assíncronas, este último cenário mencionado também se torrna pouco eficiente.

Uma arquitetura *GALS* procura incorporar as vantagens que cada tipo de implementação (síncrona, e assíncrona) podem trazer na especificação deste tipo de sistemas, quer sejam de Hardware ou de Software. Esta arquitetura oferece uma solução a situações em que uma aplicação

síncrona necessita de ler sinais de entrada assíncronos e agendar a reação antes de serem transmitidas para o programa.

# Capítulo 3 - Decomposição de Redes de Petri

## 3.1 Introdução

No que toca a tipos de controlo, existem duas formas de modelar sistemas, sendo a primeira a representação do comportamento de um sistema como um só módulo e a segunda sendo a representação do sistema decomposto em vários componentes.

A Decomposição da Rede de Petri de um modelo em várias sub-redes surge com o objetivo de facilitar a análise de redes de sistemas mais complexos. A divisão de uma rede de Petri permite a caracterização isolada do controlador de cada componente de um determinado sistema. Vários métodos de decomposição foram propostos. Muitos destes métodos surgiram com o simples propósito de facilitar a análise de um sistema, não considerando fatores tais como especificação de mecanismos de comunicação entre sub-redes e a capacidade de execução distribuída dos componentes resultantes.

## 3.2 Métodos de Decomposição de Redes de Petri

Atualmente existem vários métodos para decomposição de Redes de Petri. A maioria destes métodos procura preservar as propriedades das redes iniciais, focando-se nos facilitismos que a decomposição das redes de Petri de sistemas mais complexos oferece na análise do sistema. No entanto, a maioria das soluções não oferece suporte à capacidade de execução distribuída dos componentes gerados. Isto deve-se à inexistência de especificação de comunicação entre sub-redes nestes métodos. Muitos destes métodos consideram lugares como nós de interface de comunicação entre componentes (caso haja comunicação), levando a que seja necessária uma introdução de semântica específica adicional como suporte à capacidade de execução distribuída de um sistema.

O método proposto em [25] foca-se principalmente na perservação das propriedades após a decomposição de uma rede em várias sub-redes. Este método de decomposição baseia-se em equações lógicas. As sub-redes resultantes da aplicação deste método possuem lugares como nós de comunicação, em que ambos os lugares são duplicados do lugar da rede inicial. Um lugar de saída de uma sub-rede é um lugar de entrada de outra sub-rede.

Em [26] são aplicados métodos de decomposição e optimização como solução para um problema de agendamento geral. Tal como no método anterior, este método assume o uso de lugares de entrada e de saída nas sub-redes, em que um lugar de saída de uma sub-rede é um duplicado de um lugar de entrada de outra sub-rede. Embora diferentes, ambos os resultados gerados da aplicação dos últimos dois métodos são idênticos. Unindo-se o lugar de entrada de uma rede com o lugar de saída de outra rede é obtida a rede incial. Em ambos os últimos métodos mencionados, embora funcionais, as sub-redes resultantes não apresentam especificação sobre os mecanismos de comunicação entre os nós de interface.

Em [27] é proposto um método para execução distribuída de aplicações concorrentes. As sub-redes resultantes comunicam entre si por meio de lugares de entrada e de saída. Os lugares de entrada são representados por um círculo inscrito sobre outro de raio superior, e os lugares de saída por um triângulo inscrito num círculo. Neste método é introduzida semântica específica para os lugares de comunicação. Quando uma marca é transferida para um lugar de saída de uma sub-rede, este é transferido para o lugar de entrada da outra sub-rede e colocado em espera para consumo. Este mecanismo de comunicação descrito depende fortemente da infra-estrutura de comunicação alvo. Embora este método permita uma execução distribuída dos componentes resultantes da decomposição, a introdução de semânticas adicionais é necessária para satisfazer a comunicação descrita. Ambos os tipos de lugares de comunicação têm semânticas associadas diferentes dos outro lugares.

Em [28] é proposto um método que toma recurso a utilização de um hiper-grafo para se aplicar uma decomposição de uma rede de Petri em componentes de máquinas de estados. A comunicação entre sub-redes dá-se pela comunicação de eventos. Para esta comunicação é introduzida um lugar identificado como *NOP*, cuja sua marcação indica se a máquina de estados associada se encontra ou não activa. O disparo das transições das máquinas de estados dá-se de forma síncrona, não sendo possível a adoção de uma comunicação assíncrona dos componentes gerados.

O método apresentado em [1] baseia-se na definição de um conjunto de nós de corte da rede e na aplicação de três regras associadas aos diferentes tipos de nós definidos no conjunto mencionado.

Este método permite a decomposição da rede de um modelo global em sub-redes, oferecendo suporte à execução distribuída destas. As sub-redes geradas, dependendo da regra a aplicar, pode não manter a estrutura inicial do sistema, mas é garantido que a ordem parcial de disparo é mantida e, consequentemente, o comportamento do sistema também. O conjunto de corte definido pode conter transições e lugares, no entanto, a comunicação entre as sub-redes geradas é feita por meio de transições ligadas através de canais síncronos. Embora definidos canais síncronos na comunicação dos nós, pode-se adoptar uma solução assíncrona. Neste método, a comunicação entre componentes procura não introduzir semânticas específicas nos nós de comunicação (as semânticas para as transições de comunicação são semelhantes às semânticas do resto das transições e suportadas pelo editor da plataforma *IOPT*-Tools).

## 3.3 Método Operação *Net Splitting*

### 3.3.1 Introdução

A implementação do protótipo de uma ferramenta que aplica uma decomposição automática de redes de Petri em diversas sub-redes (*SPLIT*) baseou-se no método *Net Splitting* [1].

Neste método são tomados como principais objetivos a decomposição de uma rede de Petri em várias sub-redes de modo a que seja possível a associação de cada sub-rede ao controlador de um único componente e a implementação de comunicação entre as sub-redes geradas sem qualquer introdução de semânticas adicionais para os nós de comunicação. O resultado da aplicação pode, eventualmente, ser usado como uma preparação para uma posterior implementação e execução distribuída do sistema global por meio de comunicação assíncrona entre componentes do sistema e definição dos domínios temporais.

A comunicação entre as sub-redes geradas considerada é feita através de transições ligadas entre si por meio de canais síncronos direccionais.

Antes da etapa de decomposição da rede, primeiro é necessário definir um conjunto de corte válido, ou seja, um conjunto de lugares ou transições nos quais é possível e se pretende efectuar a operação de decomposição. Este método baseia-se na definição de um conjunto de corte válido e na aplicação de três regras para decompor a rede global. A aplicação de cada regra depende das características de cada elemento desse conjunto de corte definido. A Regra *#1* aplica-se apenas a

lugares como elemento de corte, a Regra #2 aplica-se a transições cujos seus lugares de entrada pertencem à mesma sub-rede e a Regra #3 aplica-se a transições em que os seus lugares de entrada pertencem a sub-redes diferentes.

### 3.3.2 Comunicação entre Sub-Redes no Método *Net Splitting*

Este método toma como recurso o uso de transições como nós de interface de comunicação entre sub-redes, ligadas entre si por meio de canais síncronos direccionais. A comunicação adopta o paradigma *Master/Slave* em que uma única direcção é assumida. Cada transição num conjunto de transições síncronas possui um atributo *Master* ou *Slave*. Em cada conjunto de transições síncronas, uma e apenas uma transição possui o atributo *Master* e o resto das transições do mesmo conjunto são obrigatóriamente definidas com o atributo *Slave*. Uma transição com o atributo *Slave* só dispara quando a transição *Master* do mesmo conjunto de transições síncronas dispara. Uma transição apenas pode pertencer a um único conjunto de transições síncronas. O uso de canais síncronos na comunicação entre transições foi também considerado em [29], onde são considerados dois tipos de inscrições para as transições de comunicação síncronas, *downlink* e *uplink*. As transições *uplink* podem responder a todas as transições e só se encontram habilitadas a disparar quando recebem pedido de disparo de uma transição *downlink* do mesmo canal.

Usando como referência a classe redes de Petri *IOPT*, apenas a transição com o atributo *Master* de um conjunto de transições síncronas possui a si associadas guardas de sinais e eventos.

As definições seguintes foram extraídas de [1]:

**Definição 11 (Transição com atributo).** *(De [1]) Uma transição com atributo é uma transição t com um atributo chamado t.label.*

**Definição 12 (Conjunto de transições com atributo).** *(De [1]) Um conjunto de transições com atributo é um conjunto* $LTS = T_m \cup T_s$*, onde* $T_m = \{t\}$ *e* $t.label = master$*;* $T_s$ *é um conjunto de transições com atributo onde* $\forall t' \in T_s\ t'.label = slave$ *e* $|T_s| \geq 1$*. Dado* $t \in T_m, \forall t' \in T_s \rightarrow t \neq t.$

**Definição 13 (Evento Interno).** (*De [1]) Um evento interno é um elemento que irá ser gerado com o disparar de uma transição com o atributo Master. Possui um atributo E.Master que indica a sua origem.*

**Definição 14 (Conjunto de transições síncronas).** *(De [1]) Um conjunto de transições síncronas é um tuplo* $SS = (ch, LTS, ev)$ *, onde:*

1. *ch é um identificador de canal que identifica o conjunto de transições síncronas;*
2. *LTS é um conjunto de transições etiquetadas;*
3. *ev é um evento interno gerado por* $t \in LTS.T_m$*;*

**Definição 15 (Conjunto válido de um conjunto de transições síncronas).** *(De [1])* Dado SS e SS': conjunto de transições síncronas $|SS \neq SS'$ e t : transição. Se $t \in SS.LTS \rightarrow t \notin SS'.LTS$ .

**Definição 16 (Condição de habilitação de um conjunto de transições síncronas).** *(De [1)* Seja SS um conjunto de transições síncronas e $t \in SS.LTS.T_m$. O conjunto de transições síncronas diz-se habilitado se t estiver habilitado.

**Definição 17 (Semânticas de execução do conjunto de transições síncronas).** *(De [1]) Dado SS: conjunto de transições síncronas e* $t \in SS.LTS.T_m$ *e* $t'_j \in SS.LTS.T_s$*. O conjunto de transições síncronas está habilitado se t estiver habilitado, t e* $\forall t'_j \in SS.LTS.T_s$ *disparam quando* $t'_j$ *estiver habilitado.*

**Defnição 18 (Rede de Petri *IOPT* com comunicação direccional síncrona).** *(De [1]) Uma Rede de Petri IOPT com canais de comunicação síncronos é definida como um tuplo (N,S), onde:*

1. $N = (P, T, A, TA, M, weight, weightTest, priority, isg, isg, ie, oe, osc)$ *é uma rede de Petri IOPT;*
2. *S é um conjunto de transições síncronas válido, tal que* $S = \bigcup_i SS_i$*, onde*

$SS_i = (ch_i, LTS_i, ev_i)$ *e* $\forall i(t \in SS_i.LTS.T_m \subset N.T$ *e* $N.oe(t) = N.oe(t) \wedge SS_i.ev(t))$ *e* $\forall i(t' \in SS_i.LTS.T_s \subset N.T$ *e* $.ie(t) = N.ie(t') \wedge SS_i.ev(t))$

**Definição 19 (Semânticas de Execução de uma rede *IOPT* com comunicação direccional síncrona).** *(De [1]) Dado um SS: conjunto de transições síncronas incluído numa Rede de Petri IOPT. O disparo de SS está em conformidade com o paradigma de atraso nulo, o que significa que, considerando* $t_j$ *uma transição habilitada ,*$\forall t_j \in SS.LTS.T_s$ *dispara no mesmo passo que*

*$t \in SS.LTS.T_m$. Um passo é composto por dois micro-passos, sendo o primeiro o disparo de $t \in SS.LTS.T_m$ e o segundo o disparo das $\forall t_j \in SS.LTS.T_s$.*

### 3.3.3 Definição de Conjuntos de Corte Válidos

Como já mencionado anteriormente, antes da aplicação de qualquer regra procura-se primeiro definir um conjunto de corte válido onde se poderá e irá processar a decomposição da rede.
Um conjunto de corte pode ser constituído por lugares e transições. Não existe qualquer automatismo na escolha dos elementos do conjunto de corte, sendo que este encontra-se altamente dependente da escolha do utilizador e de uma posterior verificação da validade desse conjunto. Para o conjunto de corte definido ser válido, é necessário que todos os seus elementos satisfaçam as seguintes condições:

**i)** Um elemento de um conjunto de corte não pode possuir um arco que o liga com um outro elemento do conjunto de corte;

**ii)** Pelo menos duas sub-redes isoladas têm de ser obtidas na remoção do conjunto de corte seleccionado;

**iii)** Escolhendo um lugar com várias transições de saída, como nó de corte, todas essas transições de saída deverão pertencer à mesma sub-rede. Escolhendo uma transição em situação de conflito com outras duas ou mais transições pertencentes a sub-redes diferentes, o processo de decomposição é interrompido, pois as situações de conflito não são partilháveis [1];

### 3.3.4 Regras do Método Operação *Net Splitting*

Com um conjunto de corte válido já definido, o procedimento seguinte passa pela aplicação de regras aos nós de corte. As três regras definidas em [1] obtiveram-se por meio da análise das sub-redes que podem ser obtidas na remoção de um elemento de corte, e das diversas situações identificadas em que elementos de corte válidos se podem encontrar. Contudo, a aplicação de cada regra depende das características do elemento de corte escolhido, ou seja, cada regra descreve qual o procedimento necessário para se implementar a decomposição da rede para um determinado tipo de nó que se encontra numa determinada situação. A primeira regra expõe o

procedimento necessário para a decomposição de uma rede quando o nó de corte escolhido é um lugar. As duas regras seguintes ditam o procedimento necessário para o caso de os elementos de corte serem transições, em que a segunda regra aplica-se quando os arcos de entrada têm origem em lugares que pertencem à mesma sub-rede e a terceira quando os arcos de entrada têm a sua origem em lugares que pertencem a sub-redes diferentes.

## Regra #1

A primeira regra indica o procedimento a seguir quando o elemento de corte escolhido é um lugar. Esta é a única regra aplicada aos lugares, pois apenas se considera as suas transições de entrada (que poderão pertencer a diferentes componentes) para se obter a decomposição da rede, sem qualquer interesse pelas suas transições de saída que se encontram numa situação de conflito (situações de conflito não são distribuíveis).

Num caso geral, como representado na *Figura 3.1. a)*, sendo escolhido o lugar *P3* como elemento do conjunto de corte e removendo-o da rede, considerando que as transições T1 e T2 não pertencem ao mesmo componente de alguma transição de saída do lugar de corte P3, é possível identificar-se 4 sub-redes. No entanto, as transições do conjunto de saída do lugar P3 (T3 e T4), estando numa situação de conflito, obrigatóriamente terão de se manter na mesma sub-rede após a decomposição, reduzindo este número para 3.

Após a aplicação da regra, o resultado é o observável na *Figura 3.2*. Cada transição de entrada do lugar escolhido como elemento de corte e os lugares de entrada dessas mesmas transições irão encontrar-se em sub-redes diferentes quando efectuada a decomposição. O elemento de corte (lugar P3) e as suas transições de saída (T3 e T4) permanecem na mesma sub-rede. É feita e inserida uma cópia das transições que agora pertecem a sub-redes diferentes (T1 e T2) na sub-rede que inclui o lugar de corte P3 e as transições T3 e T4.

As transições que após a decomposição se encontram, cada, em sub-redes diferentes, são ligadas por meio de canais direccionais às cópias correspondentes (transições *Slaves* do mesmo conjunto de transições síncronas) que se encontram no componente que inclui o elemento de corte. às transições T1 e T2 que se encontram em sub-redes diferentes, é declarado o atributo *Master* (T1 e T2, para a sub-rede 1 e 2 respetivamente), e às cópias correspondentes que se encontram na sub-rede 3 é declarado o atributo *Slave* do conjunto de transições síncronas respetivo. As transições

*Slave* irão encontrar-se sempre habilitadas mas disparam apenas quando a transição *Master* dispara.

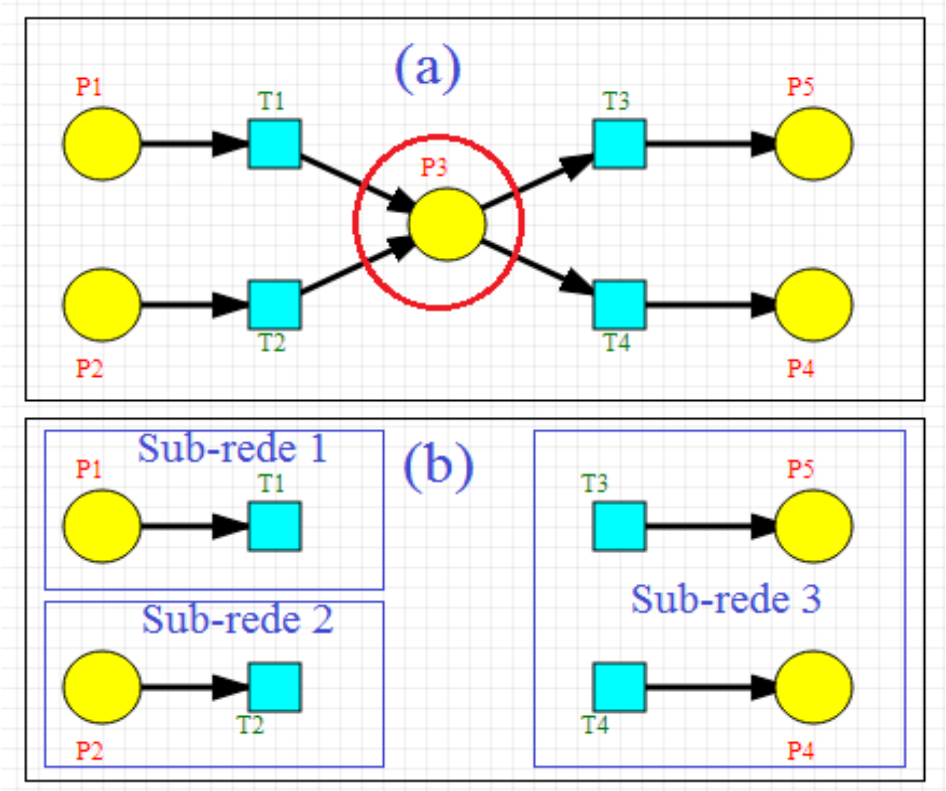


Figura 3.1 - **(adaptado de [1]).** Regra 1: Identificação e remoção do conjunto de corte.

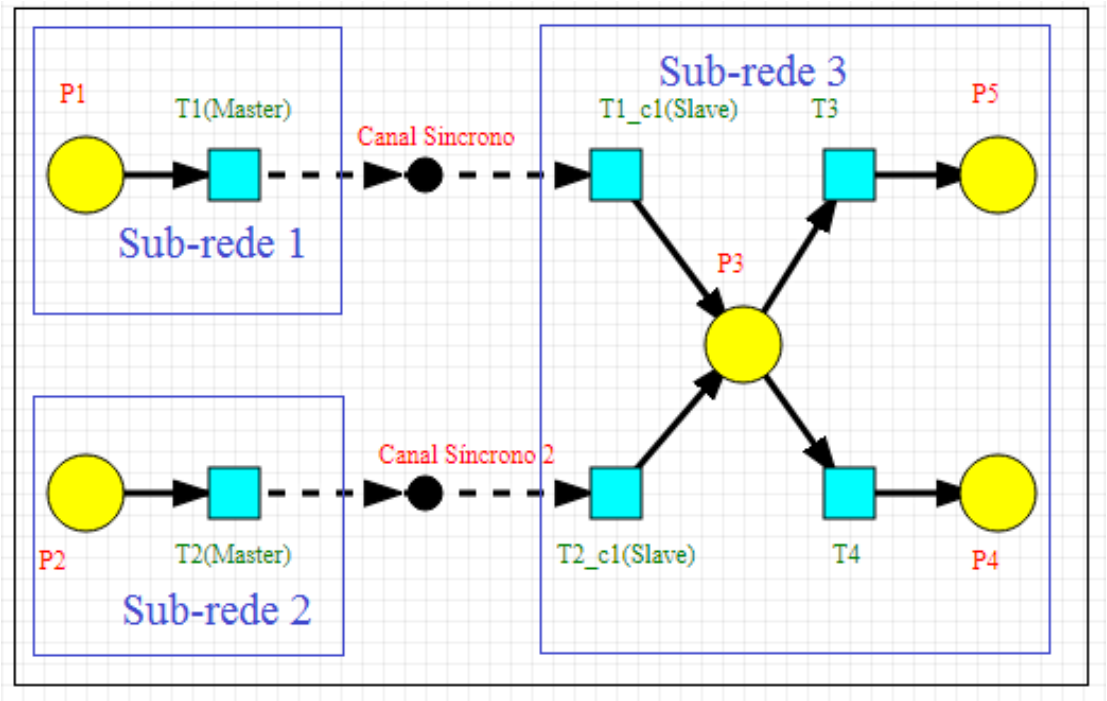


Figura 3.2 - **(adaptado de [1]).** Resultado da aplicação da Regra 1

## Regra #2

A segunda regra considera transições com lugares de entrada pertencentes à mesma sub-rede como elemento de corte e descreve o procedimento a seguir nesta situação. A transição escolhida e os seus lugares de entrada irão permanecer na mesma sub-rede, enquanto cada lugar de saída poderá pertencencer a uma outra sub-rede.

Na situação da *Figura 3.3 (a)*, removendo a transição T3 assinalada a vermelho na rede, pode-se identificar 4 redes desconexas. No entanto, considerando que T1 e T2 pertencem ao mesmo componente, este número é reduzido para 3, como representado na *Figura 3.3 (b)*. Na *Figura 3.3 (c)*, ao contrário da *Figura 3.3 (b)*, considera-se que os lugares P3 e P4 pertencem ao mesmo componente e, consequentemente, identificação de apenas duas sub-redes. Após a decomposição, na situação da *Figura 3.3 (b)*, cada lugar de saída (P3 e P4) encontrar-se-á numa sub-rede diferente, tal como ilustrado na *Figura 3.4 (a)*, sendo que a transição T3 irá permanecer na mesma sub-rede que os lugares P1 e P2. São criadas duas cópias da transição de corte T3 que são posteriormente distribuídas pelas sub-redes que possuem elementos do conjunto de saída da transição de corte seleccionada.

Já na situação da *Figura 3.3 (c)*, após a decomposição, P3 e P4 mantiveram-se na mesma sub-rede tal como ilustrado na *Figura 3.4 (b)*. Neste caso, apenas uma cópia é criada e posteriormente adicionada à sub-rede que possui elementos do conjunto de saída da transição de corte seleccionada.

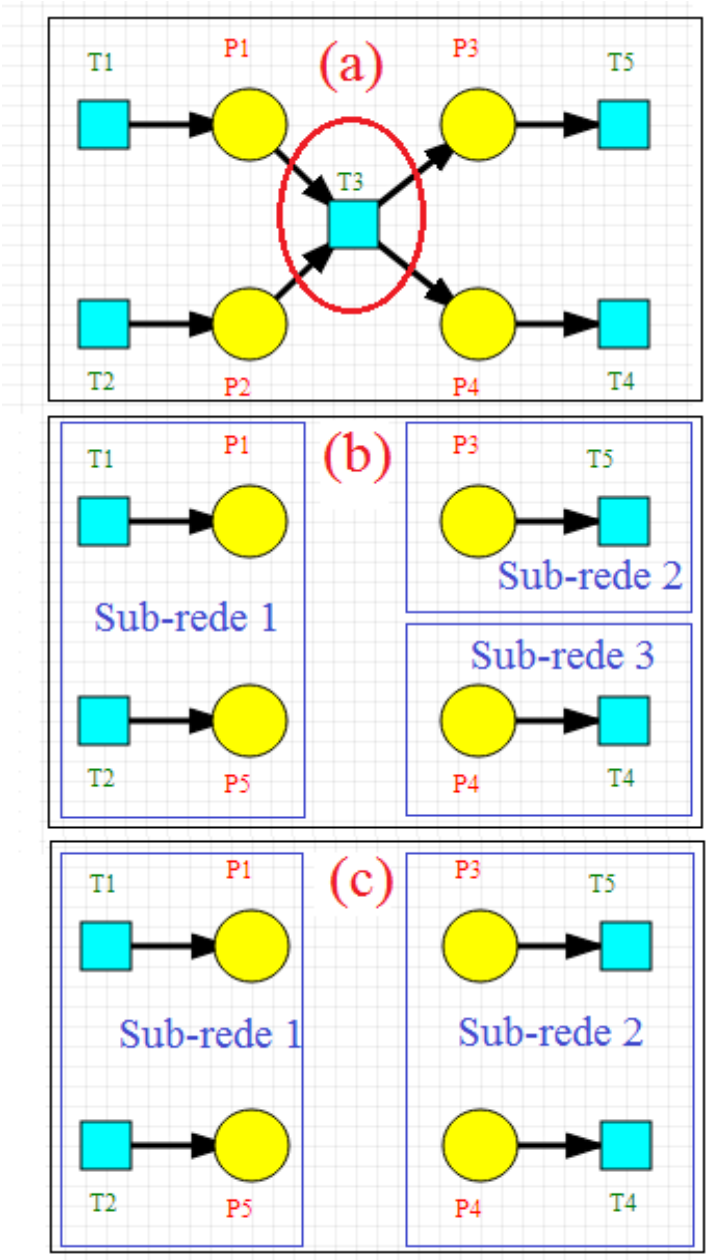


Figura 3.3 Regra 2: (a) Conjunto de corte seleccionado e remoção deste da rede; (b) P3 e P4 em diferentes componente; (c) P3 e P4 no mesmo componente;

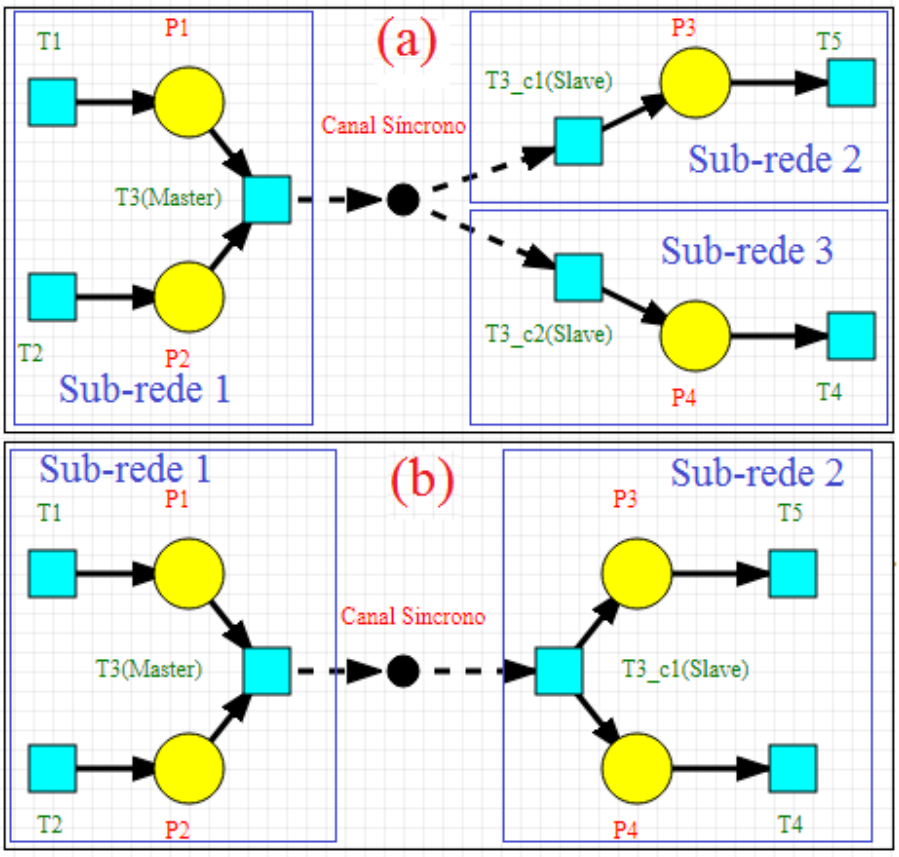


Figura 3.4 - **(adaptado de [1]).** Resultado da aplicação da Regra 2.

## Regra #3

A terceira e última regra aplica-se quando o elemento de corte escolhido é uma transição cujos lugares de entrada pertencem a diferentes componentes (sub-redes). Neste processo, os lugares de entrada ficarão, cada, em uma sub-rede diferente juntamente com uma cópia da transição de corte e com, se houver, as transições que precedem esses lugares de entrada. Na aplicação desta regra é exigido que o utilizador defina quais das cópias da transição de corte recebe o atributo *Master*. As únicas transições aptas para terem o atributo *Master* são as que pertencem a sub-redes que possuem na sua estrutura, elementos que também se encontram no conjunto de entrada da transição de corte na rede inicial.

Para este caso reaproveitou-se a rede exemplificada na Regra 2 *Figura 3.3 (a)*. A transição de corte seleccionada continua a ser a transição T3. No entanto, ambos os lugares P1 e P2 pertencem, agora, a diferentes componentes como representado na *Figura 3.5 (a) e (b)*.

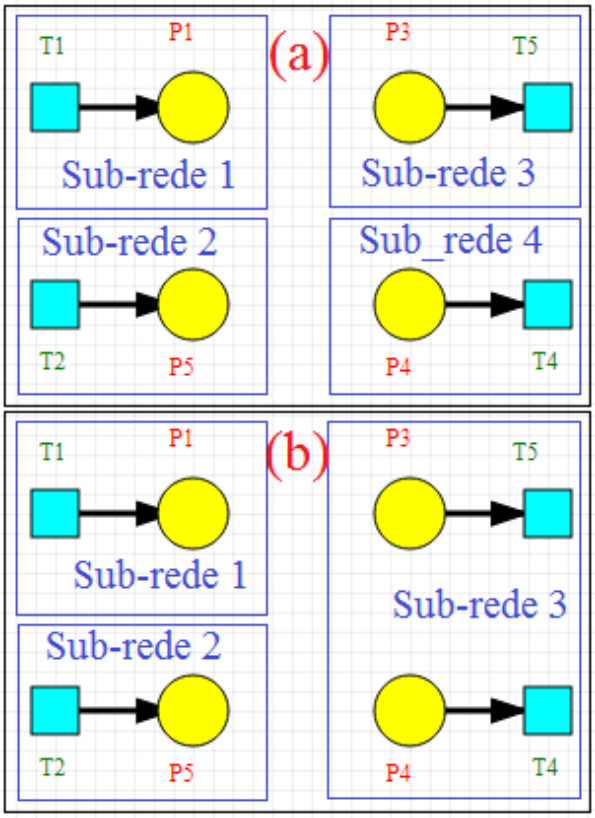


Figura 3.5 – (adaptado de [1]) –Regra 3: Remoção do conjunto de corte; (a) P3 e P4 em diferentes componentes; (b) P3 e P4 no mesmo mesmo componente;

No processo de decomposição da rede, a transição T3 com atributo *Master* é inserida numa sub-rede que possui elementos na sua estrutura que também pertencem ao conjunto de entrada da transição de corte na rede inicial (T1 ou T2, escolha do utilizador). Para o resto das sub-redes são criadas e distribuídas cópias da transição de corte com atributo *Slave*.

Nas sub-redes que possuem na sua estrutura cópias da transição de corte com atributo *Slave* e elementos no conjunto de entrada da transição de corte, como é no caso da transição *T2*, é feita uma cópia dos lugares de entrada da transição de corte e das transições que antecedem esses lugares *(T2, P2)* . Após criadas as cópias (*T2_c1(Slave)* e *P2_c1*), são adicionadas à sub-rede que possui a transição de corte com atributo *Master* (Sub-rede 1) e ligadas a esta, como é observável na *Figura 3.6 (a) e (b)*. As novas transições copiada são ligadas com as suas cópias da mesma forma que é feito com a transição de corte e as suas cópias.

O objetivo da cópia dos lugares e transições, que antecedem uma cópia *Slave* da transição de corte, é oferecer à sub-rede com a transição de corte *Master* informação sobre quando esta realmente se encontra habilitada para disparar. Caso não fossem colocadas nesta sub-rede as cópias dos elementos das outras sub-redes, a transição *Master* iria disparar mesmo que a transição *Slave* não se encontrasse habilitada, pois não teria qualquer informação sobre o estado de execução em que se encontraria a outra sub-rede.

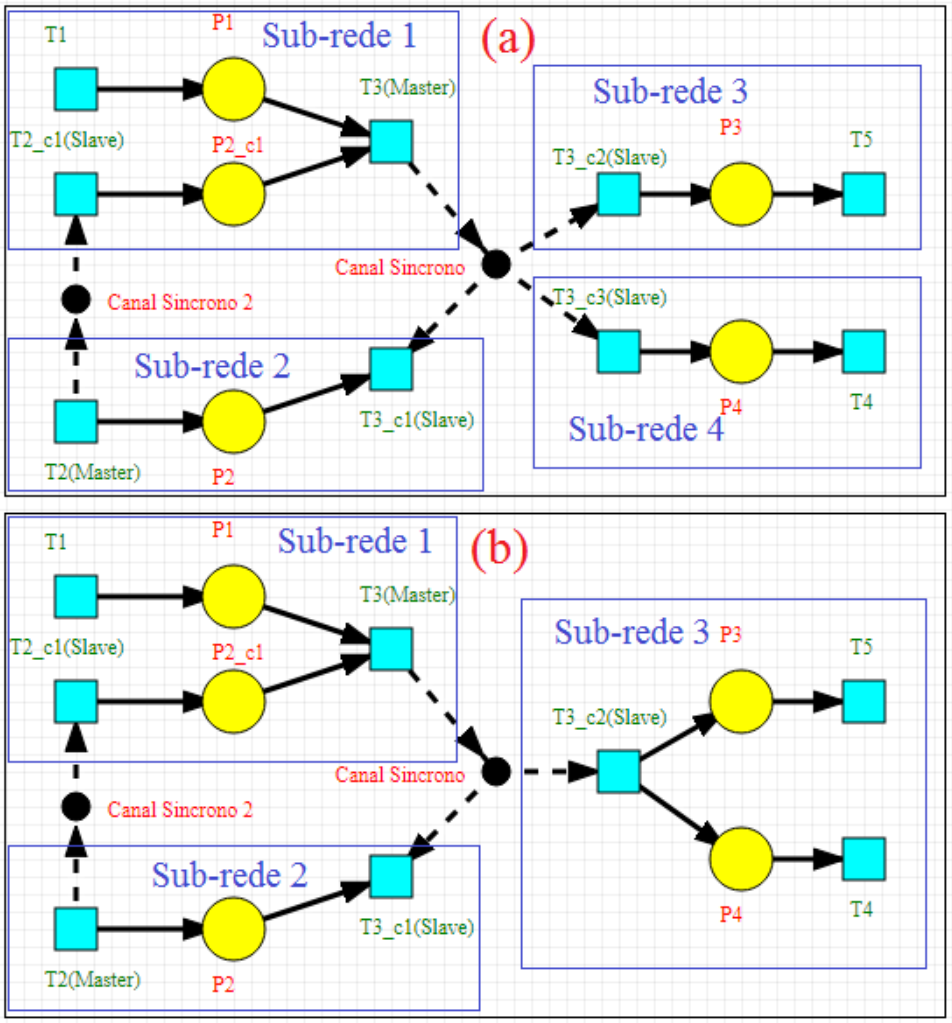


Figura 3.6 – (adaptado de [1]) – Resultado da aplicação da Regra 3;

Como já mencionado na definição de um conjunto de corte válido, selecionando a transição *T5* da *Figura 3.7 (a)* como elemento de corte e removendo-a da rede, o resultado é o representado na *Figura 3.7 (b)*. Neste caso, observa-se que a transição *T5* encontra-se numa situação de conflito com as transições *T3* e *T4*. Após a remoção de *T5*, as transições *T3* e *T4* encontram-se em sub-

redes diferentes. Após a aplicação da Regra #3 neste exemplo, o resultado é o obtido na *Figura 3.7 (c)*. Sendo que a transição *T5(Master)* localiza-se na mesma sub-rede que *T3*, é adicionada a esta mesma sub-rede uma cópia da transição *T2 (T2_c1)* e do lugar *P2 (P2_c1)*. No entanto, na sub-rede onde se encontra a transição T4 é inserida uma cópia da transição *T5 (T5_c1)*. Adoptando uma execução síncrona do sistema (transição *Master* e transições *Slaves* do mesmo conjunto de transições síncronas disparam no mesmo passo de execução), neste caso em especial, o comportamento do resultado gerado após a decomposição irá ser igual ao do modelo inicial. No entanto, quando consideramos uma execução distribuída onde não se garante o disparo da transição *Master* e das transições *Slave* do mesmo conjunto de transições síncronas no mesmo passo de execução, o disparo da transição *T5(Master)* não impossibilita o disparo da transição *T4*, o que transforma a situação de conflito entre *T4* e *T5_c1(Slave)* num processo de execução paralela, situação não desejável.

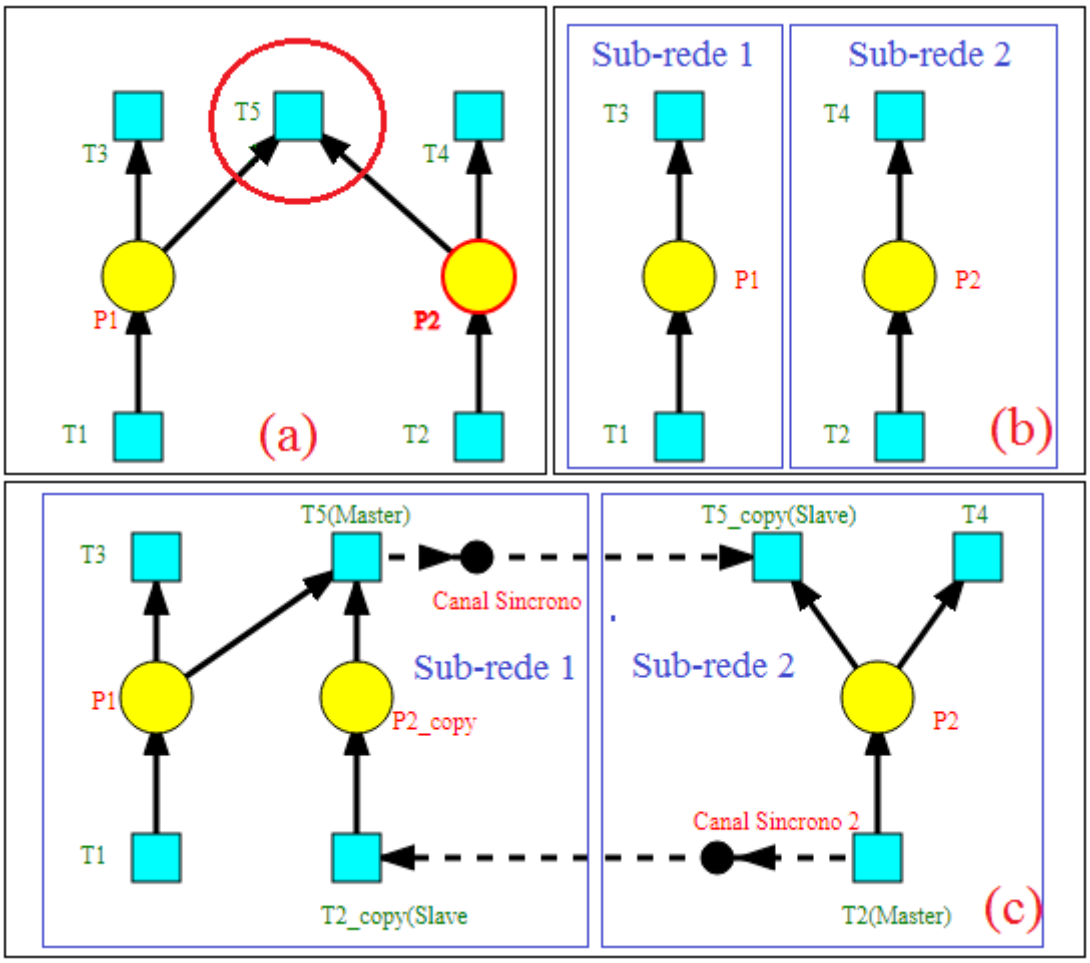


Figura 3.7 – Identificação(a), remoção do conjunto de corte(b) e aplicação da Regra 3 numa situação conflito(c);

A mesma razão se aplica no exemplo da aplicação da Regra #1 onde o lugar de corte escolhido tem de pertencer à mesma sub-rede que todas as suas transições de saída

### 3.3.5 Preservação de Propriedades

A Aplicação do método *Net Splitting* altera a estrutura inicial da rede. No entanto, é demonstrado em [1] que após a aplicação do procedimento das duas primeiras regras, "fundindo-se" as transições de um mesmo cojunto de sincronismo das sub-redes resultantes e aplicando técnicas de redução, presentes em [3], é possível obter-se a rede de Petri inicial. Relativamente às propriedades principais, é demonstrado que ambos os modelos são equivalentes.

Já no resultado da aplicação da regra #3, não é possível obter-se o segmento de rede incial com recurso a técnicas de redução, nem foi provada a preservação das propriedades., mas por observação, a sequência parcial de disparo de transições é preservada tal como o comportamento do sistema (evolução da rede).

### 3.3.6 Definições Formais

As definições formais da operação *Net Splitting* numa redes de Petri, e de todos os operadores nela usados aqui apresentadas, foram extraídos e adaptados de [1].

**Definição 20**. $A \cong B$ *significa que A é uma cópia de B mas A não possui eventos ou sinais de entrada ou saída associados que pertençam a B.*

**Definição 21**. $A = B$ *significa que A é uma cópia exacta de B.*

**Definição 22**. *A expressão* ***arc1 substituído por arc 2*** *significa que arc1 (Origem x Destino) é destruído e um novo arco com as mesmas inscrições que o arco destruído é criado com nova origem e novo destino arc2 (Novaorigem x Novodestino).*

**Definição 23.** *A expressão* ***arc1 replicado por arc2 e arc3*** *significa que o arco arc1(origem x destino) é destruído e dois novos arcos arc2 (novaorigem x novodestino) e arc3 (novaorigem' x novodestino') são criados com as mesmas inscrições que o arco destruído.*

**Definição 24 (Rede de Petri *IOPT* Decomposta).** *Uma Rede de Petri Decomposta (DIOPT) é definida como uma rede IOPT com canais de comunicação direccionais síncronos, onde uma*

*rede IOPT N é decomposta em várias sub-redes tais que* $N = \bigcup_i N_i, i \geq 2$ *onde:*

$N_j = (P', T', A', TA', M', weight', weightTest', priority', isg', ie', oe', osc')$ $j = 1,2,\ldots,i$ , ***e*** $\forall\, j \neq k$ $N_j.P' \cap N_k.P' = \emptyset$ *e* $N_j.T' \cap N_k.T' = \emptyset$ *e* $N_j.A' \cap N_k.A' = \emptyset$ *e existe um conjunto de transições síncronas SS tal, que se* $t = SS.LTS.t_m$ $t = ss.LTS.t_m \in N_j.T$ *então* $\exists t' \in N_k.T | t' \in ss.LTS.T_s$ *com* $j \neq k$.

**Definição 25.** (OPERAÇÃO *NET SPLITTING*)**.** *Dada uma Rede de Petri IOPT e um conjunto de corte CS, onde* IOPT = (P, T, A, TA, M, weight, weightTest, priority, isg, ie, oe, osc) *e* $CS = \{P', T'\}$ *e* $\forall p \in CS.P' \subseteq IOPT.P$ *e* $\forall t \in CS.T' \subseteq IOPT.T$.

*A Operação Net Splitting é definida como:* $IOPT :|: CS = DIOPT$ *, tendo DIOPT as seguintes características:*

- $(\forall p \in CS\ \exists DIOPT.IOPT_i \mid p \in DIOPT.IOPT_i.P)$ *e* $(\forall t \in {\bullet} p\ \exists DIOPT.IOPT_j | t \in DIOPT.IOPT_j.T)$. *Se* $i \neq j$ *temos:* $\exists t_{copy} \in DIOPT.IOPT_i \mid t \cong t_{copy}$ *e* $t \times p \in IOPT.F$ *é substituído por* $(t_{copy} \times p) \in DIOPT.IOPT_j.F$ *e adiciona-se* $SS_k | t = SS_k.LTS.T_m$ *e* $\forall t_{copy} \in SS_k.LTS.T_s$.

- $\forall t \in CS\ \exists DIOPT.IOPT_i \mid t \in DIOPT.IOPT_i.T$ *e* $\exists DIOPT.IOPT_j$ *com* $i \neq j$ *em que* $\exists t_{copy} | t_{copy} \cong t$ *e adiciona-se* $SS_l | t = SS_l.LTS.T_m$ *e* $\forall t_{copy} \in SS_l.LTS.T_s$ *e os seguintes axiomas:*

  -$\forall p \in {\bullet} t$ *se* $p \in DIOPT.IOPT_j.P$ *temos:*
  $\exists p_{copy} \in DIOPT.IOPT_j.P \mid p_{copy} = p$ *e* $(p \times t) \in IOPT.F$ *é replicada por:*$\{(p_{copy} \times t) \in DIOPT.IOPT_j.F\}$;
  *e* $\forall t' \in {\bullet} p$ *temos:*
  $\exists t'_{copy} \in DIOPT.IOPT_i \mid t'_{copy} \cong t'$ *e* $(t' \times p) \in IOPT.F$ *é replicada por:*
  $(t'_{copy} \times p_{copy}) \in IOPT_i.F \cup (t' \times p) \in IOPT_j.F$ *e adiciona-se* $SS_n | t' = SS_n.LTS.T_m$ *e* $\forall t'_{copy} \in SS_n.LTS.T_s$.

  -$\forall p' \in t {\bullet} |$ *se* $p' \in DIOPT.IOPT_j.P$ *,então* $(t \times p') \in IOPT.F$ *é substituído por* $(t_{copy} \times p') \in DIOPT.IOPT_j.F$.

- *Todosos outros lugares, transições e arcos da rede mantêm-se idênticos aos da rede IOPT inicial;*

### 3.3.7 Exemplo de Aplicação e Considerações

Considerando novamente a rede relativa ao controlador do sistema constituído por duas vagonetes representado pela *Figura 2.5*, é preparada uma decomposição desta mesma rede cujo conjunto de corte definido é constituído pelas transições assinaladas a vermelho na *Figura 3.8 a)* (transições: GO e BACK). Removendo os elementos de corte (*GO* e *BACK*) identificam-se as seguintes quatro redes isoladas representadas na *Figura 3.8. b)*.

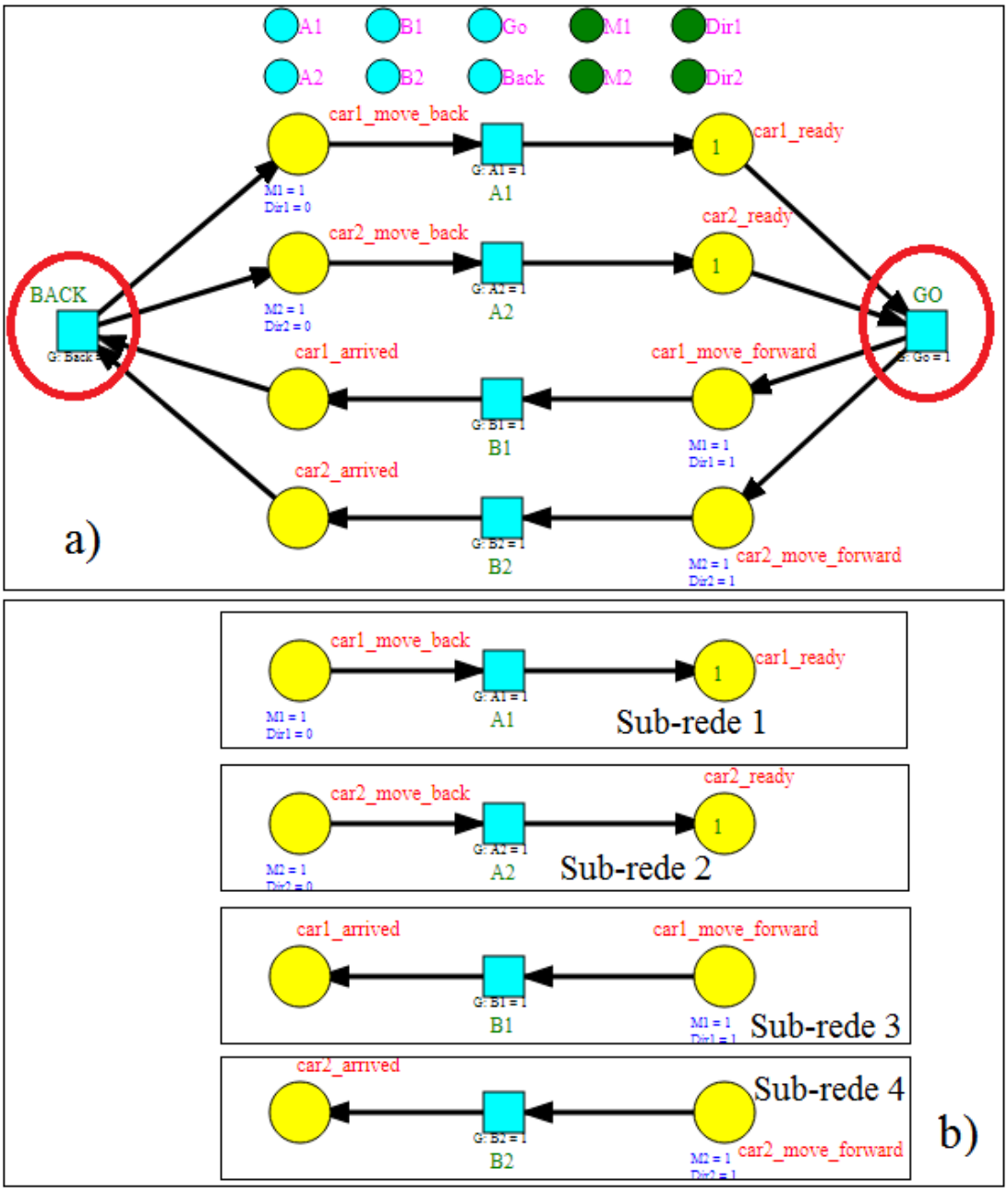


Figura 3.8 – Identificação (a) e remoção do conjunto de corte (b) da rede do sistema de duas vagonetes;

Sendo que, os elementos de corte são transições com lugares de entrada pertencentes a diferentes sub-redes, a regra do método *Net Splitting* a aplicar será a terceira. Na aplicação desta regra, o *designer* tem a liberdade de escolher em qual dos componentes a gerar ficam as transições do conjunto de transições síncronas *GO* e *BACK* com o atributo *Master*.
O resultado da aplicação da Regra #3 encontra-se representado na *Figura 3.9.*

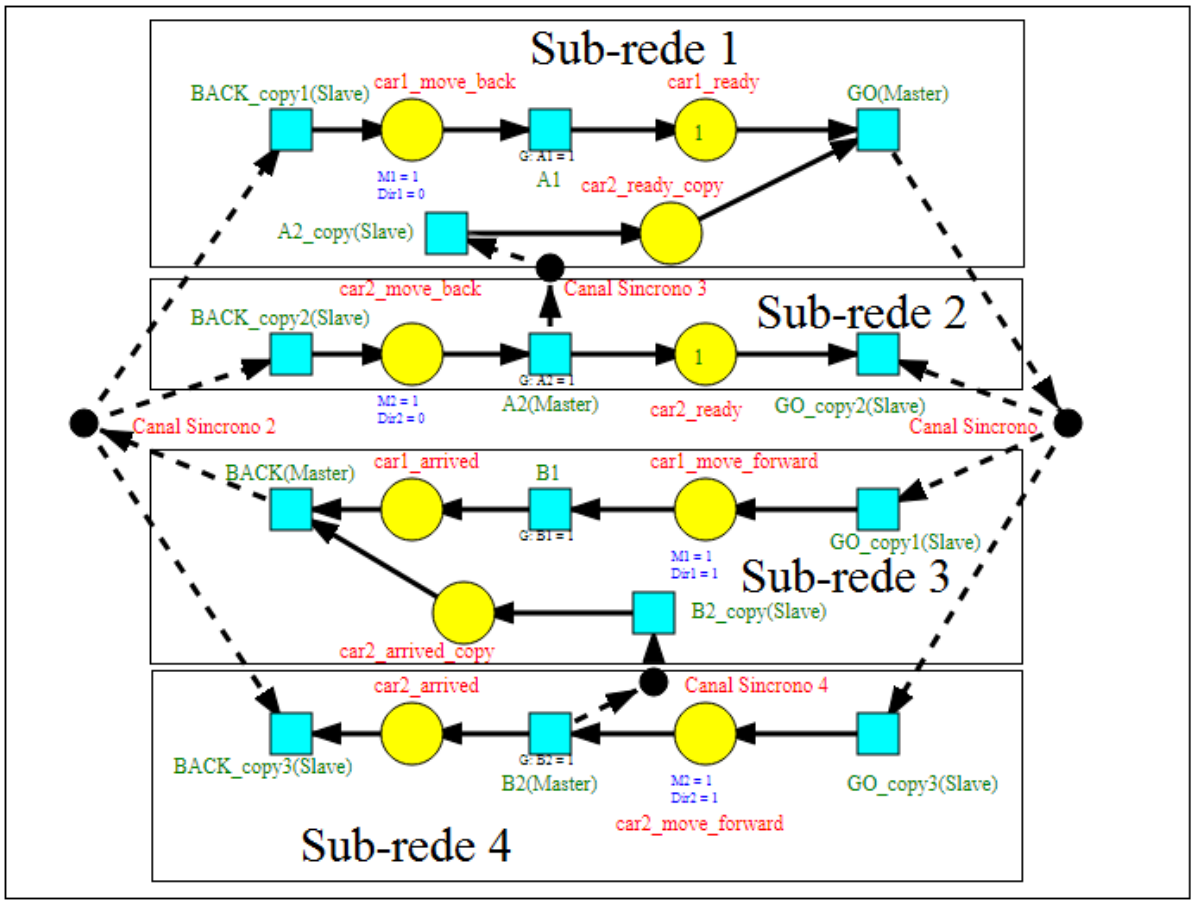


Figura 3.9 - Resultado da aplicação da Regra 3 do método operação *Net Splitting* no sistema de duas vagonetes.

O número de componentes que se pretende obter deste sistema são os controladores de cada uma das duas vagonetes (apenas dois componentes). No entanto, a aplicação da regra 3 irá gerar quatro componentes (como representado na *Figura 3.9*). Um componente relativo à ordem de deslocamento da vagonete 1 do ponto de carga ao ponto de descarga. outro componente para a ordem de deslocamento da vagonete 1 do ponto de descarga de volta ao ponto de carga e outros 2 componentes idênticos a estes últimos descritos mas associados ao deslocamento da vagonete 2.

Para obtenção de apenas dois componentes, cada um relativo a cada vagonete, é necessário ter em consideração a definição de um conjunto de pares de elementos que o desenvolvedor pretende que se mantenham na mesma sub-rede. Para isto, e considerando os pares de elementos que se pretende manter na mesma sub-rede, *A1* e *B1* para um par e *A2* e *B2* para o segundo par, obtemos o resultado expresso na rede da *Figura 3.10*. Neste caso, o resultado obtido para cada componente inclui, na sua estrutura, o deslocamento do ponto de carga ao ponto de descarga e o deslocamento inverso, respetivo a cada vagonete.

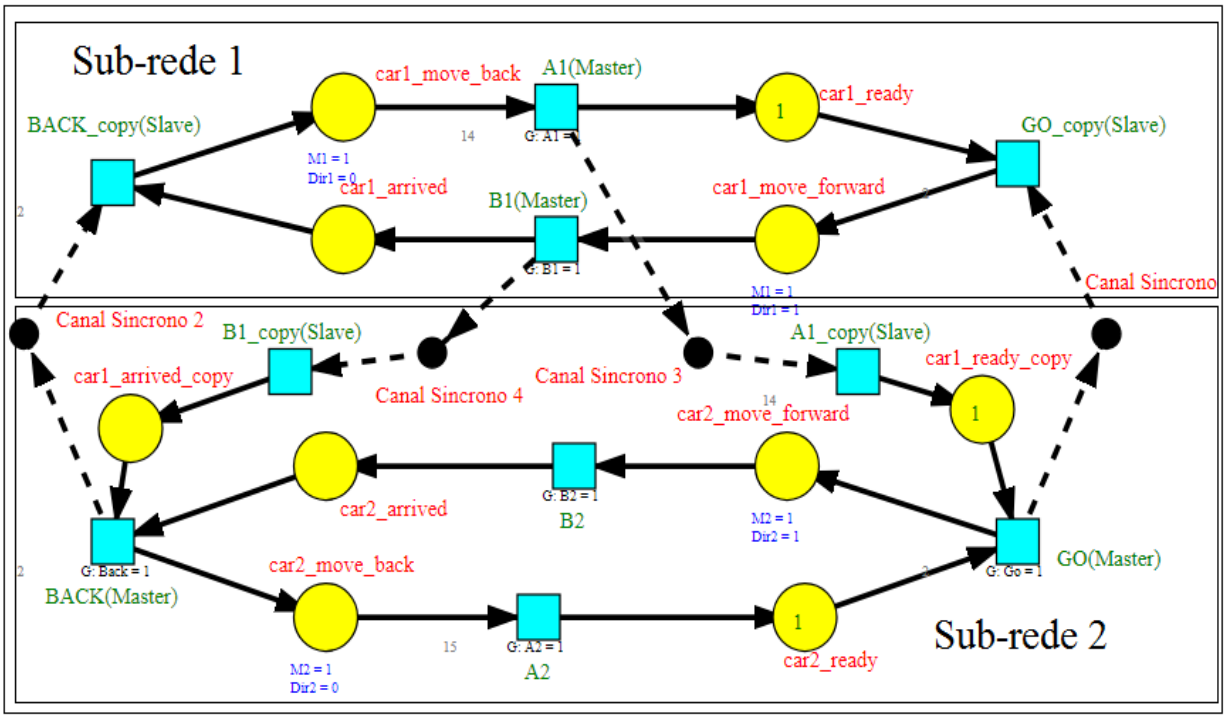


Figura 3.10 - Resultado obtido sendo definido que os elementos A1 e A2, B1 e B2 necessitam manter-se na mesma rede.

# Capítulo 4 - Implementação

## 4.1 Recursos

### 4.1.1 Ficheiros *PNML*

Na implementação da ferramenta *SPLIT* recorreu-se aos modelos *PNML* gerados, a partir da plataforma *IOPT-Tools*, como ficheiros alvo de leitura e teste. Nestes ficheiros encontra-se toda a informação necessária sobre a rede referente ao modelo inicial e parâmetros definidos pelo utilizador, tais como: elementos seleccionados do conjunto de corte; pares de elementos que se pretende que fiquem na mesma sub-rede; localização da transição *Master*, caso o elemento de corte seleccionado seja uma transição com lugares de entrada pertencentes a diferentes sub-redes e que não se encontre numa situação de conflito impossível à decomposição.

*PNML* é o formato para intercâmbio de redes de Petri baseado em *XML* [30]. O seu principal propóstio é oferecer uma organização hierárquica descritiva de dados e permitir uma fácil leitura e transporte desta mesma informação por parte de outras aplicações ou sistemas. Inúmeros sistemas apresentam incompatibilidades na forma como os seus dados são formatados, o que resulta em tempo perdido na tarefa de conversão de dados quando há necessidade de partilha destes entre os sistemas referidos. Ambas as linguagens de marcação mencionadas guardam informação em simples ficheiros de texto, oferecendo uma forma de armazenamento de dados independente de elementos de *Hardware* ou *Software*, o que resulta na simplificação da partilha, disponibilidade e transporte de dados entre plataformas.

Na *Figura 4.1* é representado de forma simplificada como se encontra organizado um ficheiro *PNML* de uma rede *IOPT*. No Interior dos ficheiros *PNML* encontra-se toda a informação sobre a rede definida no editor (*IOPT - Tools*). Dentro do nó *net* (nó da rede) encontram-se nós com informação sobre sinais e eventos de entrada e de saída (nós *input* e *output* da *Figura 4.1*, respetivamente) e sobre todos os elementos da rede (nós *place, transition* e *arc*). Todos os elementos possuem identificadores únicos (*id*). No caso dos arcos, além do seu identificador, possuem também os identificadores dos nós onde as suas extremidades se encontram ligadas, *source* para o nó de origem e *target* para o nó de incidência.

```
<?xml version="1.0" encoding="iso-8859-1" standalone="no"?>
<Snoopy revision="0" version="0">
    <pnml>
        <net id="1" name="..." type="IOPT">
            <input>...</input>
            <output>...</output>
            ...
            <place id = "1">
                <name>...</name>
                <comment>...</comment>
                <initialMarking></initialMarking>
                <bound></bound>
                <graphics></graphics>
                <signalOutputActions></signalOutputActions>
                <timedomain>...</timedomain>
            </place>
            <transition id = "2">
                <name>...</name>
                <comment>...</comment>
                <priority>...</priority>
                <signalInputGuards>...</signalInputGuards>
                <graphics>...</graphics>
                <signalOutputActions>...</signalOutputActions>
                <outputEvents>...</outputEvents>
                <timedomain>...</timedomain>
            </transition>
            <arc id = "3" source "2" target = "1">
                <type>normal</type>
                <graphics>...</graphics>
                <inscription>...</inscription>
            </arc>
        </net>
   </pnml>
</Snoopy>
```

Figura 4.1 – Ficheiro *PNML* de uma rede *IOPT*.

Tal como presente nas definições da classe *IOPT* de redes de Petri, aos lugares (nós *place*) encontra-se associado: a marcação inicial (nó *initialMarking*); o atributo físico (nó *bound*); e acções associadas a sinais de saída. Às transições (nós *transition*) encontram-se associados: guardas de sinais de entrada, eventos de entrada e de saída (nós *signalInputGuards*, *inputEvents* e *outputEvents*, respetivamente); e prioridade (nó *priotrity*). Nos arcos encontra-se associada informação sobre a sua inscrição ou peso (nó *inscription*) e sobre o seu tipo (nó *type*). Em ambos os lugares e transições encontra-se associada informação sobre domínios de tempo (nó *timedomain*).

Na *Figura 4.2 (a)* e *(b)* encontra-se a representação textual dos canais assíncronos e síncronos. Na aplicação da ferramenta *SPLIT* existe a necessidade de criação de nós de canais síncronos e assíncronos entre conjuntos de transições síncronos. Deste modo, torna-se importante saber como estes se encontram representados textualmente para que seja então possível criar os canais a partir do programa C desenvolvido. No editor da plataforma *IOPT*-Tools ambos os canais síncronos são

um tipo especial de lugar com indicação sobre o seu tipo (nó *placetype*) e do tipo de canal que representam (nó *actype*).

(a)

```
<place id="1">
    <name>
      <text>Canal Assíncrono</text>
      ...
    </name>
    <comment>....</comment>
    <placetype>asyncchannel</placetype>
    <actype>simple</actype>
    <graphics>...</graphics>
  </place>
```

(b)

```
<place id="2">
  <name>
    <text>Canal Sínncrono</text>
    ...
  </name>
  <comment>...</comment>
  <placetype>asyncchannel</placetype>
  <actype>syncset</actype>
  <graphics>
    <position page="1" x="210" y="400"/>
  </graphics>
</place>
```

Figura 4.2 – Representação textual dos Canais Assíncronos (a) e Síncronos (b) da plataforma *IOPT*-Tools.

### 4.1.2 Linguagem de Programação C

A Linguagem de Programação C é das linguagens de baixo nível mais utilizadas. Esta linguagem de programação foi criada por Dennis Ritchie, entre 1969 e 1973, para o desenvolvimento de sistemas operativos. Por tratar-se de uma linguagem de baixo nível, ou seja, a compilação dos programas ocorre em níveis próximos do *Hardware*, a execução de programas C torna-se extremamente rápida. Devido à baixa utilização e acesso a endereços de memória de dispositivos, esta linguagem é também muito utilizada em programação de *Hardware*, nomeadamente em dispositivos onde recursos de processamento e memória são extremamente limitados, tais como microcontroladores, *Arduinos*, entre outros. A linguagem C adopta os paradigmas, imperativo e de estrutura. Em documentos C é especificada uma sequência de instruções, iterações e condições de modo a alcançar-se um determinado estado do programa. Todo o código desenvolvido encontra-se distribuído em sub-rotinas (ou funções) que podem comunicar entre si por meio de parâmetros de entrada e de saída.

Todos os processos referentes ao Algoritmo do método *SPLIT* foram implementados utilizando a linguagem de programação C. Uma das principais razões que levaram ao uso desta linguagem é

a existência de bibliotecas de funções desenvolvidas nesta mesma linguagem que oferecem suporte à leitura e manipulação de ficheiros *XML* e *PNML*.

A biblioteca usada para este fim, *Libxml2* [31], trata-se de um conjunto de ferramentas para análise, leitura e escrita de ficheiros *XML* desenvolvido em C para o projecto *GNOME*, que se trata de um completo conjunto de aplicações e ferramentas de ambiente de trabalho baseado inteiramente em software gratuito sob a linceça do *MIT*.

## 4.2 Integração com a Plataforma *IOPT-Tools*

Para integração do programa C desenvolvido na Plataforma *IOPT*-Tools, implementou-se o *plug-in SPLIT* representado na *Figura 4.3*. Este é responsável pelo pedido e leitura dos parâmetros inseridos no editor pelo utilizador, como, por exemplo, o conjunto de corte seleccionado (nó *param name = "SelNodes"*). O conteúdo seleccionado no editor lido pelo *plug-in* é então impresso no nó *pnml* do ficheiro *PNML* (pelo editor). Após a evocação do *plug-in* e da definição dos parâmetros por parte do utilizador, o ficheiro *PNML* do modelo em desenvolvimento é encaminhado para o executável do Programa C desenvolvido. Na definição dos parâmetros, quando evocado o *plug-in*, também é guardado no ficheiro *PNML* o tipo de implementação da comunicação entre os componentes gerados, Sícrona ou Assíncrona (nó *param name = ImpType*).

```
<?xml version="1.0"?>
<?xml-stylesheet href="show-plugin-params.xsl" type="html" charset="utf-8"?>
<pnml-editor-plugin name="SPLIT">
   <code path="plugins/test_plug.sh" />
   <text-info>...</text-info>
   <param name="ImpType" type="enum" default="0">
       <option value="0">Synchronous</option>
       <option value="1">Assynchronous</option>
   </param>
   <param name="SelNodes" type="selection">
       <node-type>place</node-type>
       <node-type>transition</node-type>
   </param>
</pnml-editor-plugin>
```

Figura 4.3 – *Plug-in SPLIT* do editor da plataforma *IOPT-Tools*.

No ficheiro *PNML* da rede, a informação do *plug-in* evocado encontra-se representada na *Figura 4.4.* Os elementos de corte seleccionados no editor irão ter um nó no plug-in com o seu tipo e identificador único da rede (*id*).

```
<?xml version="1.0" encoding="iso-8859-1" standalone="no"?>
<Snoopy revision="0" version="0">
  <pnml>
      <net id="1" name="..." type="IOPT">
      ...
      </net>
      <pnml-editor-plugin name="test plugin nr. 2">
          <code path="plugins/test_plug.sh"/>
          <param name="ImpType" type="enum" default="0" value="1"/>
          <param name="SelNodes" type="selection">
               <node type="transition" id="..."/>
               <node type="place" id="..."/>
          </param>
    </pnml-editor-plugin>
  </pnml>
</Snoopy>
```

Figura 4.4 – Exemplo de um Ficheiro *PNML* após execução do *plug-in SPLIT*.

## 4.3 Implementação do Programa C Desenvolvido

### 4.3.1 Estrutura de Dados

Na *Figura 4.5* encontram-se representadas as estruturas de dados C, utilizadas para implementação da ferramenta *SPLIT*. A forma como as estruturas C foram definidas pretende satisfazer relações com os conjuntos definidos no método de decomposição em que se baseia a implementação da ferramenta *SPLIT*. A informação presente nos campos das estruturas tem o objetivo de organizar toda a informação necessária para ser possível criar um novo ficheiro *PNML* com o resultado da decomposição da rede. Ao longo deste excerto é feita uma breve descrição sobre cada estrutura.

```
struct element
{
  char *id;
  char *type;
  char *preset[100];
  char *postset[100];
  char *NP_MLoc;
  char *reachability[100];
  int subnet_Loc;
  char *inscription;
  int copies;
};
```

```
struct cutting_elem {
    char* id;
    char* p_conflict[100];
};
```

Figura 4.5 – Estruturas de dados C definidas para os elementos da rede inicial (a) e para os elementos do conjunto de corte (b);

Para cada elemento da rede inicial é criada uma estrutura *element* (representada na *Figura 4.5)*, em que cada estrutura tem o objetivo de armazenar e disponibilizar informação de um único elemento da rede inicial do ficheiro *PNML*.

- *id* – Identificador único do elemento da rede do ficheiro *PNML*;

- *type* – Tipo do elemento, lugar, transição ou arco;

- *preset* – Vetor análogo ao conjunto de entrada do elemento da rede;

- *postset* - Vetor análogo ao conjunto de saída do elemento da rede;

- *pairs_MLoc* – Guarda o identificador do elemento par deste elemento (se definido).

- *reachability* – Vetor para se guardar o identificador único de todos os elementos da rede que são alcançáveis a partir do elemento a que a estrutura pertence (Só é definido para lugares e transições que têm no seu conjunto de entrada um elemento do conjunto de corte);

- *subnet_Loc* – Identificador inteiro único de cada sub-rede gerada para indicar a qual sub-rede irá pertencer este elemento (Só é definido para lugares e transições que têm no seu conjunto de entrada um elemento de corte);

- *inscription* – Valor do peso de um arco (Só é definido para arcos);

- *copies* – Indicador do número de cópias deste elemento a serem criadas no novo ficheiro *PNML* após a decomposição;

É criada uma estrutura *cutting_elem (*representada na *Figura 4.5*) para cada elemento do conjunto de corte seleccionado, de modo a ser possível identificar as estruturas *element* dos elementos de corte.

- *id* – identificador único do elemento na rede do ficheiro PNML;

- *p_conflict* – Guarda identificador dos lugares de entrada que possuem mais do que uma transição de saída (Preenchido só em transições de corte com lugares de entrada pertencentes a diferentes sub-redes;

Foram definidas mais duas estruturas, representadas na *Figura 4.6*, que são criadas e preenchidas no decorrer da execução do programa implementado.

```
struct subnet
{
    char *id;
    char *nodes[100];
    int subnet_id;
    char *cs_nodes[50];
    char *output_nodes[50];
};
```

```
struct Lts_copy
{
    char *id;
    char *type;
    char *New_postset[50];
    char *New_preset[50];
    int new_cut;
    int atribute;
    char *copies_id[50];
};
```

Figura 4.6 – Estruturas de dados C definidas para as sub-redes (a) e para os novos elementos da rede decomposta e conjuntos de transições síncronas (b);

A estrutura *subnet* tem o objetivo de, após removidos os elementos de corte da rede inicial, identificar e armazenar informação necessária sobre cada sub-rede gerada (antes e após a aplicação das regras do método).

- *id* – Identificador único da sub-rede no Vetor de estruturas Subnets;

- *subnet_id* – Identificador único inteiro da sub-rede;

- *nodes* – Vetor que representa o conjunto dos elementos da rede inicial que pertencem a esta sub-rede

- *cs_nodes* – Vetor que representa o conjunto dos elementos de conjunto de corte que têm ligações, na rede inicial, com elementos que pertencem a esta rede;

- *output_nodes* – Vetor utilizado para guardar o identificador das transições de corte *Master* que possuem uma transição *Slave* nesta sub-rede e elementos que pertencem ao seu conjunto de entrada na rede inicial.

A estrutura *Lts_copy* tem o objetivo de guardar informação sobre elementos da rede inicial que foram alvo de alteração e precisam de ser copiados, cópias de elementos que estão ainda por ser inseridos na rede e conjuntos de transições síncronas.

- *id* – Identificador único do elemento na rede inicial;

- *type* – Tipo do elemento (lugar, transição, arcos);

- *New_postset* – Vetor que representa o novo conjunto de saída do elemento no modelo da rede decomposta;

- *New_preset* – Vetor que representa o novo conjunto de entrada do elemento no modelo da rede decomposta;

- *new_cut* – Para se identificar as transições dentro do Vetor de estruturas *Lts_copies*, resultantes da aplicação das Regras #1 e #3 mas que não fazem parte do conjunto de corte.

- *atribute* – Inteiro, para se identificar quais dos elementos desta estrutura são originais da rede inicial (*atribute* = 1, no caso das transições identifica transições *Master*) e quais são cópias (*atribute* = 0, no caso das transições identifica transições *Slaves*).

- *copies_id* – Para guardar o identificador de todos os outros elementos com estrutura *Lts_copy* criada, que são cópias deste elemento. Para transições *Master*, irá ter o identificador de todas as suas cópias de forma análoga ao conjunto de transições síncronas. Para transições *Slave* irá conter apenas o identificador da transição *Master* respetiva.

### 4.3.2 Algoritmo Implementado

O fluxo sequêncial da *Figura 4.7* representa o Algoritmo principal implementado no programa C desenvolvido (***SPLIT.c***). Este Algoritmo foi adaptado do proposto em [1].

Numa primeira fase é lido o ficheiro *PNML* enviado pelo editor. Desta leitura guarda-se os dados da rede inicial relevantes à operação de decomposição da rede *IOPT*.

A fase de validação do Conjunto de Corte definido encontra-se dividida em três etapas. Esta divisão foi feita de acordo com a ordem de adição de informação relevante nas estruturas para realização de cada etapa.

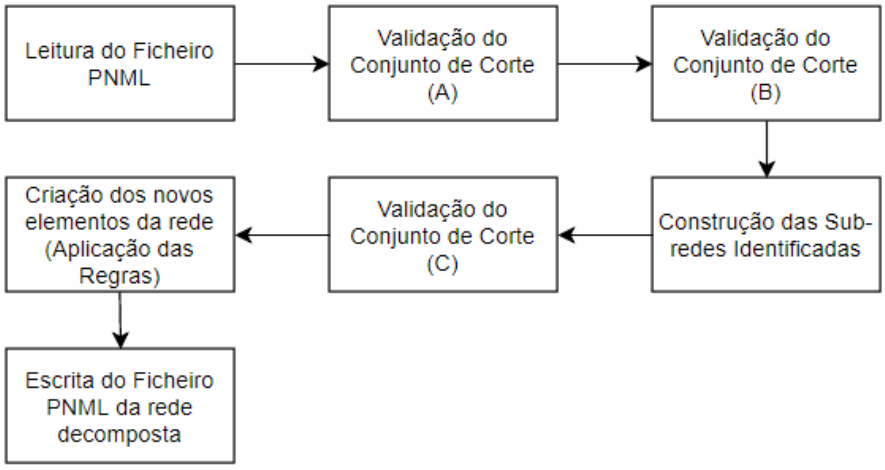


Figura 4.7 – Fluxo do algoritmo implementado no programa C desenvolvido.

- Validação do Conjunto de Corte (A) – Verifica na rede inicial se existe alguma ligação entre elementos do conjunto de corte.

- Validação do Conjunto de Corte (B) – Verifica se o número de redes isoladas geradas com a remoção do conjunto de corte da rede inicial é inferior a dois. Verifica se o Conjunto dos pares definido é válido.

- Validação do Conjunto de Corte (C) – Verifica se existe transições no conjunto de corte em situação de conflito impossível à aplicação da decomposição. Verifica se a localização de cada transição *Master* definida é válida.

Nas etapas da criação das sub-redes geradas e criação de novos elementos da rede e conjuntos de transições síncronas resultantes da aplicação das regras do método *Net Splitting*, é calculada e disponibilizada informação sobre cada sub-rede e cópias de elementos a criar no novo ficheiro *PNML*.

Numa etapa final, com base na informação disponibilizada nas estruturas das cópias de elementos e de conjuntos de transições síncronas, procedeu-se à criação do novo ficheiro *PNML* com a representação da nova rede decomposta.

### Leitura do Ficheiro PNML

Para leitura dos ficheiros *PNML* recorreu-se às funções das bibliotecas *Libxml2*. Efectua-se uma análise do ficheiro. É Inicializado um apontador para percorrer as estruturas dos nós do ficheiro *PNML*, de modo a ser possível a extracção de informação de cada nó.

*(Linhas 4 -16)* - Durante a leitura de cada nó relativo a um elemento da rede (nós *place*, *transition* e *arc*), é criada uma estrutura *element*. Duma primeira leitura de cada nó da rede são extraído s apenas os valores do tipo de nó, identificador único na rede inicial (*id*) e o conteúdo do texto do nó *Comment*, caso haja, para os lugares e transições. O conteúdo do nó *Comment* é guardado no campo *pairs_Mloc* da estrutura *element*.

| ***Algoritmo*** *– Leitura do Ficheiro PNML* |
|---|
| Parâmetros Relevantes:<br>- Apontador para uma estrutura de nós XML;<br>- Vetor de estruturas de elementos da rede inicial *element - Net_elements*; |

```
1:  Para cada Nó no interior do Nó raíz do Ficheiro PNML faz {
2:      Se nó = "pnml" então {
3:          Para cada Nó no interior do Nó "pnml" faz {
4:              Se Nó = "net" então {
5:                  Para cada Nó no interior do nó "net" faz {
6:                      Se Nó = "place" então {
7:                          - Cria estrutura element do Lugar;
8:                      }
9:                      Se Nó = "transition" então {
10:                         - Cria estrutura element da Transição;
11:                     }
12:                     Se Nó = "transition" então {
13:                             - Cria estrutura element da Transição;
14:                     }
15:                 }
16:             }
17:             Se Nó = "pnml-editor-plugin" então {
18:                 Para cada Nó no interior do Nó "pnml-editor-plugin" faz {
19:                     Se Nó = "param" e valor do nó name = "SelNodes" então {
20:                         Para cada Nó no interior do Nó "param" faz {
21:                             - Cria estrutura cutting_elem do elemento de Corte;
22:                         }
23:                     }
24:                 }
25:             }
26:         }
27:     }
28: }
```

(Linhas 12 – 14) - Para os arcos é também extraída informação sobre quais os nós (lugar e transição) que este arco liga, e sobre o seu peso (valor do nó *inscription* dos nós dos arcos no ficheiro PNML). É adicionado nos campos *preset* e *postset* da estrutura *element*, de cada arco, os valores que se encontram nos identificadores do seu nó no ficheiro *PNML*, *source* e *target*, respetivamente. É guardado em *inscription* da estrutura *element* do arco o valor do seu peso.

(Linhas 17 – 28) - Durante a leitura do ficheiro *PNML*, além da extracção de dados relativos a cada elemento da rede, também é criada uma estrutura *cutting_elem* para cada elemento de corte identificado no nó do *plug-in*.

### Validação do Conjunto de Corte (A)

(Linhas 1 – 12) - Tendo os identificadores dos elementos a que os arcos se encontram ligados nos campos da estrutura *element*, *preset* e *postset* de cada arco, procedeu-se ao preenchimento dos mesmos campos das estruturas *element* das transições e lugares da rede inicial. No exemplo de um arco que tem origem num lugar e incide numa transição, é colocado o identificador da transição no campo *postset* da estrutura *element* do lugar, e o identificador do lugar no campo *preset* da transição. Desta forma, é possível ter-se acesso directo, na estrutura *element* de cada lugar e transição, aos identificadores dos elementos que pertencem aos seus conjuntos de entrada e de saída na rede inicial.

(Linhas 13 – 24) - Para cada elemento da rede inicial com estrutura *cutting_elem* criada, é verificado na estrutura *element* respetiva se existe no seu conjunto de entrada e saída (*preset* e *postset*) o identificador de um elemento da rede que também tenha estrutura *cutting_elem* criada. Ou seja, é verificado se existem ligações na rede inicial entre elementos do conjunto de corte definido. Se existirem, o programa é interrompido e é retornada a mensagem de erro a indicar que o conjunto de corte definido é inválido.

***Algoritmo*** *– Validação do Conjunto de Corte(A);*

Parãmetros Relevantes:
- Vetor de estruturas de elementos da rede inicial, *element* – Net_elements;
- Vetor de estruturas de elementos de corte *cutting_elem* – *Cutting_set*;

```
1:  Para cada posição x de Net_elements faz {
2:      Se (Net_elements[x].type = "arc") então {
3:          Para cada posição y em Net_elements faz {
4:              Se (Net_elements [x].postset[0] == Net_elements [y].id) então {
5:                  Adiciona a Net_elements [y].preset o valor de Net_elements[x].preset;
6:              }
7:              Se (Net_elements [x].preset[0] == Net_elements [y].id) então {
8:                 Adiciona a Net_elements [y].preset o valor de Net_elements[x].postset;
9:              }
10:         }
11:     }
12: }
13: Para cada posição x de Cutting_set faz {
14:     Para cada posição y de Net_elements faz{
15:         Se (Cutting_set[x].id == Net_elements[y].id) então {
16:             Se ( Existe k e i, tal que (Net_elements[y].preset[k] == Cutting_set[i].id ) ) então {
17:                 Erro: Conjunto de Corte Inválido;
18:             }
19:             Se ( Existe k e i, tal que (Net_elements[y].postset[k] == Cutting_set[i].id ) ) então {
20:                 Erro: Conjunto de Corte Inválido;
21:             }
22:         }
23:     }
24: }
```

### Validação do Conjunto de Corte (B)

Este Algoritmo descreve o procedimento efectuado para o cálculo do número de sub-redes geradas após a decomposição. Para se evitar uma representação demasiado extensa deste Algoritmo, este encontra-se dividido em três sub-Algoritmos. O primeiro descreve o procedimento efectuado na leitura e validação do conjunto de pares de elementos da mesma sub-rede. O segundo sub-Algoritmo descreve o procedimento efectuado para o cálculo do número de sub-redes geradas com a remoção dos elementos do conjunto de corte da rede inicial. O terceiro representa a contagem de sub-redes geradas e criação das estruturas *subnet* para cada sub-rede identificada.

### Sub-Algoritmo 1: Validação do Conjunto de Corte (B)

Linhas (1 – 10) -Neste Algoritmo, é primeiro identificado os elementos de cada par de elementos que ficarão na mesma sub-rede após a decomposição. Elementos que possuem na sua estrutura da rede inicial, *element,* um valor não nulo em *NP_Mloc* e não pertencem ao conjunto de corte, são identificados como o primeiro elemento de um par de elementos. O identificador guardado em *NP_Mloc* pertence ao segundo elemento do par correspondente. Os identificadores dos pares de elementos são guardados num novo Vetor *NetPairs*. Cada posição par dentro deste Vetor encontra-se o identificador do primeiro elemento de um par e na posição ímpar imediatamente acima, o identificador do segundo elemento do par respetivo.

***Sub - Algoritmo 1*** *– Validação do Conjunto de Corte(B);*

Parâmetros Relevantes:
- Vetor de estruturas de elementos da rede inicial, element - *Net_elements*;
- Vetor de estruturas de elementos de corte, *cutting_elem* - *Cutting_set*;
- Vetor de pares de elementos da mesma sub-rede – *NetPairs*;

```
1:  n = 0;
2:  Para cada posição x em Net_elements faz {
3:      Se ( Valor em Net_elements[x].NP_Mloc não é nulo) então {
4:          Se Não Existe posição y em Cutting_set, tal que Net_elements[x].id = Cutting_set[y].id então {
5:              - NetPairs[n] = Net_elements[x].id
6:              - NetPairs[n+1] = Net_elements[x].NP_Mloc;
7:              - n = n + 2;
8:          }
9:      }
10: }
11: Para cada n em NetPairs faz {
12:     Se Não Existe uma posição x em Net_elements, tal que NetPairs[n] = Net_elements[x].id então {
13:         - Erro: Conjunto de  Pares de Elementos Seleccionados Inválido;
15:     }
16: }
```

(Linhas 11 – 16) – Para cada identificador único dos elementos dos pares definidos, é verificada a sua existência na rede inicial. Caso não exista um elemento com este identificador, a decomposição da rede é interrompida e é retornada uma mensagem de erro indicando que o conjunto dos pares de elementos definido é inválido.

## Sub-Algoritmo 2: Validação do Conjunto de Corte (B)

***Sub - Algoritmo 2*** *– Validação do Conjunto de Corte (B)*

Parâmetros Relevantes:
- Vetor de estruturas de elementos da rede inicial, element - Net_elements;
- Vetor de estruturas de elementos de corte, cutting_elem - Cutting_set;
- Vetor de pares de elementos da mesma sub-rede – NetPairs;

```
1:  k = 1; (Para atribuir identificadores inteiros únicos a ss_node)
2:  Para cada x  em Net_elements faz {
3:      Para todos i e n, tal que Net_elements[x].preset[ i ] = Cutting_set[ n ].id  faz {
4:          Ignorando o Conjunto de Corte e Elementos Já Existentes em Net_elements[x].reachability faz {
5:              - Adicionar a Net_elements[x].reachability nova alcançabilidade;
6:              - Net_elemets[x].subnet_loc = k;
7:              - k = k +1;
8:          }
9:      }
10: }
11: Para cada posição x  em Net_elements, tal que Net_elements[x].subnet_loc ≠ 0  faz {
12:     Para cada posição y em Net_elements, tal que Net_elements[x].subnet_loc ≠ 0 faz {
13:         Se (Existe n e k, tal que Net_elements[x].reachability[n] = Net_elements[y].reachability[k]) então {
14:             - Substitui o valor em Net_elements[y].subnet_loc por Net_elements[x].subnet_loc;
15:         }
16:     }
17: }
18: n = 0;
19: Enquanto NetPairs[n] tiver valor Não Nulo faz {
20:     Para cada posição x em Net_elements, tal que Net_elements[x].subnet_Loc ≠ 0  faz {
21:         Se (Existe k, tal que Net_elements[x].reachability[k] = NetPairs[n]) então {
22:             Para cada posição y em Net_elements, tal que Net_elements[y].subnet_Loc ≠ 0 faz {
23:                 Se (Existe i , tal que Net_elements[y].reachability[i] = NetPairs[n + 1]) então {
24:                     Para todos z, tal que Net_elements[z].sub_Loc = Net_elements[y].sub_Loc faz {
25:                         - Substituir o valor em Net_elements{z].sub_Loc por Net_elements[x].sub_Loc;
26                      }
27:                 }
28:             }
29:         }
30:     }
31:     n = n + 2; (Passa para a posição de NetPairs Onde Se Encontra o Primeiro Elemento do Próximo Par).
32: }
33: Para cada posição x em Net_elements, tal que Net_elements[x].type = Lugar faz  {
34:     Para cada posição y em Cutting_set, tal que Net_elements[x].id = Cutting_set[y]  faz {
35:         Para cada posição z, tal que Net_elements[x].postset[0] = Net_elements[z].id  faz  {
36:             Para todos w e n > 1 tal que Net_elements[x].postset[n] = Net_elements[w].id  faz {
37:                 - Substitui o valor em Net_elements[w].subnet_Loc por Net_elemens[z].id;
38:             }
39:         }
40:     }
41: }
```

(Linhas 1 – 10) - Procurou-se pelas estruturas de todos os elementos da rede inicial que possuem no seu conjunto de entrada (*preset)* um elemento contido no conjunto de corte definido (com estrutura *cutting_elem* criada), ou que possuam um conjunto de entrada vazio. Para estes elementos é guardado no seu conjunto de alcançabilidade (representado pelo Vetor *reachability* da sua estrutura C de elementos da rede inicial), os identificadores únicos de todos os elementos do seu conjunto de saída e de todos os elementos do conjunto de saída dos elementos do seu conjunto de saida, recursivamente. Esta recursividade é realizada até ser encontrado, no conjunto de saída de um elemento, o identificador de um elemento do conjunto de corte ou de um elemento já com identificador existente no conjunto de alcançabilidade (em caso de ciclos) ou com conjunto de saída vazio. É também atribuído, na estrutura de cada elemento da rede inicial com conjunto de alcançabilidade não nulo, um valor inteiro único que é guardado em *subnet_Loc.*

(Linhas 11 - 17) - Este processo tem o objetivo de indicar quais elementos que pertencem à mesma sub-rede. Para cada elemento que tem na sua estrutura de elementos da rede inicial valor *subnet_Loc* diferente de zero são comparados os seus conjuntos de alcançabilidade. Se ambos os elementos possuem identificadores de elementos no seu campo *reachability* (alcançabilidade) em comum, então o valor do *subnet_Loc* do segundo elemento é substituído pelo mesmo valor do primeiro elemento de comparação. Estruturas de elementos da rede inicial com valores *subnet_Loc* diferente de zero, iguais, irão pertencer à mesma sub-rede, juntamente com todos os elementos identificados nos seus conjuntos de alcançabilidade.

(Linhas 18 - 32) - Para os pares de elementos da mesma sub-rede o processo é semelhante. Para cada par identificado em *NetPairs* é procurada a estrutura do elemento da rede inicial que tem no seu campo reachability o identificador do primeiro elemento do par, de modo a obter-se o seu indicador de localização *subnet_Loc*. De seguida, para cada primeiro elemento de cada par, é procurada a estrutura do elemento que possui o identificador do segundo elemento do par, de modo a obter-se o seu indicador de localização. Finalizando, em todas as estruturas de elementos que possui um indicador de localização *subnet_Loc* idêntico ao do elemento que possui o segundo elemento do par no seu conjunto de de alcançabilidade, é substituido esse valor pelo indicador de localização da estrutura do elemento que possui no seu conjunto de alcançabilidade o identificador do primeiro elemento do par.

(Linhas 33 – 41) - Quando um lugar é seleccionado como elemento de corte, removendo-o da rede, todas as transições do seu conjunto de saída devem manter-se na mesma sub-rede. Para cada lugar do conjunto de corte (se houver), é procurada as estruturas das transições que pertencem ao seu conjunto de saída. Se as estruturas de todas as transições do conjunto de saída do lugar de corte tiverem um indicador de localização *subnet_Loc* diferente, então o indicador de localização destas transições, se diferentes, são igualadas ao indicador de localização da primeira transição

identificada no conjunto de saída do lugar de corte.

### Sub-Algoritmo 3: Validação do Conjunto de Corte (B)

(Linhas 1 – 7) - Para cada estrutura de elementos com indicador de localização *subnet_Loc* diferente de zero, é verificado se existe alguma estrutura de uma sub-rede criada com identificador único de sub-redes *subnet_id* igual. Se não existir, é criada uma nova estrutura de sub-rede com identificador *subnet_id* igual ao indicador de localização do elemento da rede. O número de estruturas de sub-redes criadas é contado.

(Linhas 8 – 10) - Caso o número de sub-redes resultantes da remoção do conjunto de corte da rede inicial seja inferior a dois, o programa é interrompido e é retornada uma mensagem de erro a indicar que o conjunto de corte seleccionado é inválido.

| ***Sub-Algoritmo 3*** *- Validação do Conjunto de Corte (B)* |
|---|
| Parâmetros Relevantes:<br>- Vetor de estruturas de elementos da rede inicial, *element - Net_elements*;<br>- Vetor de estruturas de de Sub-redes, *subnet – Subnets*; |
| 1: Número de Sub-Redes = 0; (Inicialização de um contador de Sub-Redes);<br>2: Para cada posição x em Net_elements, tal que Net_elements[x].subnet_Loc ≠ 0 faz {<br>3: Se Não Existe posição y em Subnets, tal que Net_elements[x].subnet_Loc = Subnets[y].subnet_id faz {<br>4: - Cria estrutura subnet de uma Sub-Rede com subnet_id = Net_elements[x].subnet_Loc;<br>5: - Número de Sub-Redes = Número de Sub-Redes + 1;<br>6: }<br>7: }<br>8: Se (Número de Sub-redes < 2) então {<br>9: Erro: Conjunto de Corte Inválido!<br>10: } |

## Construção das Sub-redes geradas

(Linhas 1 - 9) - Para cada elemento da rede inicial que possui na sua estrutura (*element*) o identificador *subnet_Loc* diferente de zero, é colocado no conjunto de elementos da estrutura da sub-rede, com identificador inteiro *subnet_id* idêntico, todos os identificadores de elementos contidos no seu conjunto de alcançabilidade (*reachability*). Identificadores que já existam no conjunto dos elementos da sub-rede não são novamente adicionados.

***Algoritmo** - Construção das redes isoladas geradas;*

Parâmetros Relevantes:
- Vetor de estruturas de elementos da rede inicial, *element - Net_elements*;
- Vetor de estruturas de elementos de corte, *cutting_elem - Cutting_set*;
- Vetor de estruturas de Sub-redes, *subnet – Subnets*;

```
1:  Para cada posição x em Net_elements, tal que (Net_elements[x].ss_node != 0)  faz {
2:      Para cada posição y em Subnets, tal que Net_elements[x].ss_nodes = Subnets[y].ss_nodes_s  faz {
3:          Para cada k em Net_elements[x].reachability[k]  faz {
4:              Se (Ainda não existe em Subnets[y].nodes o valor de Net_elements[x].reachability[k])  então {
5:                  - Adicionar a Subnets[y].nodes o valor em Net_elements[x].reachability[k];
6:              }
7:          }
8:      }
9:  }
10: Para cada posição x em Subnets faz {
11:     Para cada posição y em Cutting_set faz  {
12:         Para cada posição z em Net_elements, tal que Net_elements[z].id = Cutting_set[y].id  faz  {
13:             Se (Existe n e k, tal que Subnets[x].nodes[n] = Net_elements[k].preset[n])  então  {
14:                 - Adiciona a Submets[x].cs_nodes o valor de Cutting_set[y].id;
15:                 Se (Net_elements[z].type = transição então {
15:                     - Adiciona a Subnets[x].output_nodes o valor de Cutting_set[y].id;
16:                 }
17:             }
18:             Se (Existe n e k, tal que Subnets[x].nodes[n] = Net_elements[k].postset [n])  então  {
19:                 Se (Ainda Não Existe em Subnets[x].cs_nodes o valor de Cutting_set[y].id)  então {
20:                     - Adiciona a Subnets[x].cs_nodes o valor em Cutting_set[y].id;
21:                 }
22:             }
23:         }
24:     }
25: }
```

(Linhas 10 – 24) Para cada estrutura de sub-redes, é preenchido o conjunto de elementos de corte intervenientes (cs_nodes) com o identificador de todos os elementos de corte que na rede inicial possuem ligações com elementes dessa mesma sub-rede. Caso o elemento do conjunto de corte seja uma transição que possui elementos de determinada sub-rede no seu conjunto de entrada da rede inicial, então é adicionado o seu identificador ao campo *output_nodes* da estrutura da sub-rede. Esta última operação serve para indicar que nesta rede se pode localizar a transição *Master* da transição de corte.

### Validação do Conjunto de Corte (C)

O Algoritmo seguinte encontra-se dividido em dois sub-Algoritmos sequênciais. O primeiro descreve o procedimento implementado para verificação da existência de transições no conjunto de corte seleccionado que na rede inicial se encontram numa situação de conflito impossível à realização da decomposição. A segunda descreve o processo de validação das localizações das transições *Master* definidas.

### Sub-Algoritmo 1: Validação do Conjunto de Corte (C)

***Sub - Algoritmo 1*** – *Validação do Conjunto de Corte (C)*

Parâmetros Relevantes:
- Vetor de estruturas de elementos da rede inicial, *element - Net_elements;*
- Vetor de estruturas de elementos de corte, *cutting_elem - Cutting_set*;
- Vetor de estruturas de Sub-redes, *subnet – Subnets*;

```
1:  Para cada posição x em Cutting_set faz {
2:      Para cada posição y em Net_elements faz {
1:          Se (Cutting_set[x].id = Net_elements[y].id e Net_elements[y].type = transição)  faz {
4:              Para cada n em  Net_elements[y].preset[n] faz {
5:                  Para cada z em Net_elements, tal que Net_elements[z].id = Net_elements[y].preset[n] faz {
6:                      Se (Net_elements[z].postset tem o identificador de mais do que uma transição) então {
7:                          - Adiciona a Cutting_set[x].p_conflict o identificador em Net_elements[z].id;
8:                      }
9;                  }
10:             }
11:         }
12:     }
13: }
14: Para cada posição x em Cutting_set faz {
15:     Se (Cutting_set[x].p_conflict tem mais do que um elemento) então {
16:         Para cada posição y em Subnets {
17:             Se (Existe em Subnets[y].nodes o id de Cutting_set[x].p_conflict[0] (1º elemento)) então  {
18:                 Se (Existe n > 0 tal que Cutting_set[x].p_conflict[n] não existe em Subnets[y].nodes) então {
19:                     - Error: Conjunto de Corte Inválido;
20:                 }
21:             }
22:         }
23:     }
24: }
```

(Linhas 1 – 13) - Para cada estrutura da rede inicial de transições identificadas no conjunto de corte é verificado se os seus lugares de entrada na rede inicial possuem nas suas estruturas *elemento*, respetivas, mais do que dois elementos identificados no seu conjunto de saída.

(Linhas 14 – 24) - Caso afirmativo, e se houver mais do que um lugar com estas características, então para todos estes lugares é verificado se os seus identificadores se encontram no conjunto de elementos da mesma estrutura de sub-redes. Se não estiverem, o programa é interrompido e é retornada mensagem de erro a indicar que o conjunto de corte selecionado é inválido.

### Sub-Algoritmo 2: Validação do Conjunto de Corte (C)

(Linhas 1 – 10) -Elementos que na sua estrutura de elementos da rede inicial apresentam um valor não nulo em *NP_MLoc*, são transições de corte com lugares de entrada pertencentes a diferentes sub-redes. Os identificadores únicos da rede inicial das transições de corte (com estrutura *cutting_elem*) são filtradas para um novo Vetor *T_Cutting_set*. O valor *NP_Mloc* da estrutura de elementos da rede inicial da transição de corte é guardado num Vetor *Master_Loc*, na mesma posição em que o seu identificador se encontra no vetor *T_Cutting_set*. O valor presente em *NP_MLoc* de transições de corte é o identificador de um elemento da rede inicial. Este último elemento pertence à sub-rede onde se pretende que se localize a transição *Master* respetiva.

***Sub - Algoritmo 2*** *– Validação do Conjunto de Corte (C)*

Parâmetros Relevantes:
- Vetor de estruturas de elementos da rede inicial, *element - Net_elements;*
- Vetor de estruturas de elementos de corte, *cutting_elem - Cutting_set*;
- Vetor de estruturas de Sub-redes, *subnet – Subnets*;
- Vetor com identificadores de transições de corte – *T_Cutting_set;*
- Vetor com identificadores de elementos de localização da transição *Master – Master_Loc*;

```
1:  n = 0;
2:  Para cada posição x em Cutting_set faz {
3:      Para cada posição y em Net_elements tal que Cutting_set[x].id = Net_elements[y].id faz {
4:          Se (Net_elements[y].type = transição) então {
5:              - T_Cutting_set[n] = Net_elements[y].id;
6:              - Master_Loc[n] = Net_elements[y].NP_MLoc;
7:              - n = n + 1;
8:          }
9:      }
10: }
11: Para cada posição x em Subnets faz {
12:     Para cada posição y em T_Cutting_set faz {
13:         Se (Master_Loc[y] ≠ "0"e Existe em Subnets[x].nodes) então {
14:             Se (Não existe T_Cutting_set[y] em Subnet[x}.cs_nodes) então {
15:                 - Erro: Localização da Transição Master Definida Inválida;
16:             }
17:         }
18:     }
19: }
```

(Linhas 11 – 19) – Verifica-se, para cada transição de corte (identificada em cada posição do Vetor *T_Cutting_set*), se o elemento identificado na mesma posição do Vetor *Master_loc* encontra-se também identificado no conjunto de elementos de uma sub-rede que não tenha o identificador da transição de corte, respetiva, no seu conjunto de elementos de corte com ligações a elementos desta sub-rede. Caso não tenha, é retornada uma mensagem de erro a indicar que a localização definida da transição *Master* de uma transição de corte é inválida,

### Criação dos novos elementos da rede resultantes da aplicação das Regras do Método *Net Splitting*

A etapa de criação dos elementos resultantes da aplicação das regras processa-se com o preenchimento das estruturas *Lts_copy* dos elementos da rede inicial que foram copiados, e das suas cópias ainda por se criar no ficheiro *PNML*.

O Algoritmo encontra-se dividido em três sub-Algoritmos. O primeiro descreve o processo de criação das cópias de elementos resultantes da aplicação da regra 1 do método *Net Splitting*, ou seja, transições de entrada do lugar de corte que não pertencem à mesma sub-rede que o lugar de corte após a decomposição.

O segundo sub-Algoritmo descreve o processo de obtenção de informação sobre a criação das cópias das transições de corte resultantes da aplicação da Regra #2 e #3.

O terceiro descreve o procedimento na obtenção de informação sobre a criação das cópias adicionais de elementos da rede inicial, resultantes da aplicação da Regra 3. Ou seja, quando uma transição possui lugares de entrada na rede inicial pertencentes a diferentes sub-redes.

Toda a informação guardada nas estruturas *Lts_copy* é posteriormente utilizada como auxílio para criação dos nós de novos elementos da rede a adicionar no ficheiro *PNML*.

### Sub-Algoritmo 1: Criação dos novos elementos da rede resultantes da aplicação das Regras do Método Net Splitting

Para o caso dos lugares de corte, não há necessidade de qualquer cópia pois a comunicação entre sub-redes dá-se por meio de transições. No entanto, é necessário criar cópias de transições que se encontram no seu conjunto de entrada.

(Linhas 1 – 6) - Na estrutura de cada sub-rede é verificado se os elementos de corte contidos no conjunto de elementos de corte que possuem ligações com elementos da sub-rede (cs_nodes) são

lugares. Caso sejam lugares, verifica-se se existem elementos dessa sub-rede que são elementos de saída do lugar de corte na rede inicial. Caso afirmativo, é criada uma nova estrutura *Lts_copy* para o lugar de corte.

(Linha 5 ) - Na criação da estrutura *Lts_copy* do lugar de corte, é preenchido seu novo conjunto de saída (campo *New_postset*) com os identificadores dos elementos do conjunto de saída do lugar de corte na rede inicial. O campo do novo conjunto de entrada (*New_preset*) da estrutura *Lts_copy* do lugar de corte, é preenchido com o identificador das transições que pertencem ao conjunto de entrada do lugar de corte na rede inicial e que também pertencem à mesma sub-rede que os elementos contidos no conjunto de saída do lugar de corte na rede inicial. O campo *atribute* é preenchido com valor 1 (Lugar original da rede inicial). O identificador único da estrutura *Lts_copy* do lugar de corte é o seu próprio identificador único na rede inicial e o seu tipo *place*.

***Sub-Algoritmo 1*** *– Criação das cópias dos elementos resultantes da aplicação da Regra 1*

Parâmetros Relevantes:
- Vetor de estruturas de elementos da rede inicial, element - Net_elements;
- Vetor de estruturas de Sub-redes, subnet – Subnets;

```
1:  Para cada posição x em Subnets faz {
2:      Para cada posição y em Net_elements, tal que Net_elements[y].type = "place" faz {
3:          Se Existe n, tal que Subnets[x].cs_nodes[n] = Net_elements[y].id faz {
4:              Se Existe um k e um i, tal que Net_elements[y].postset[k] = Subnet[y].nodes[i] então {
5:                  - Cria nova estrutura em lts_copies do Lugar de Corte:
6:              }
7:              Se (Não existe um k e um i, tal que Net_elements[y].postset[k] = Subnet[y].nodes[i] então {
8:                  Para todos h e j, tal que Net_elements[y].preset[k] = Subnet[y].nodes[i] faz {
9:                      - Cria estrutura Lts_copy da transição identificada em Subnets[y].nodes[i]: (Master)
10:                     - Cria estrutura Lts_copy da cópia da transição identificada em Subnets[y].nodes[i]: (Slave)
11:                 }
12:             }
13:         }
14:     }
15: }
```

(Linhas 7 – 14) - Caso só exista, nessa sub-rede, transições contidas no conjunto de entrada do lugar de corte na rede inicial, é criada uma estrutura *Lts_copy* para cada transição e para a cópia da transição.

(Linha 9) - Na criação da estrutura *Lts_copy* de cada transição a ser copiada, é preenchido o campo *id* com o identificador único da transição na rede inicial, o tipo com *transition*, o campo *atribute* é colocado a 1 de modo a indicar que é uma transição *Master* de um conjunto de transições

síncronas. É também adicionado ao conjunto das cópias da estrutura Lts_copy de cada transição (*copies_id*) um novo identificador único criado que é posteriormente atribuído à sua cópia. O novo conjunto de saída é preenchido com o identificador dos elementos que pertencem ao seu conjunto de saída na rede inicial com a excepção do identificador do lugar de corte. O seu novo conjunto de entrada é preenchido com o identificador de todos os elementos que pertencem ao seu conjunto de entrada na rede inicial, caso esses elementos não sejam elementos do conjunto de corte.

(Linha 10) – Na criação da estrutura *Lts_copy* da cópia de cada transição, é preenchido o seu *id* com o novo identificador único na rede criado no passo anterior. O seu tipo é preenchido com *transition* e o valor *atribute* é colocado a 0 de modo a indicar que é uma transição *Slave* de um conjunto de transições síncronas. No conjunto de cópias na estrutura *Lts_copy* deste elemente é colocado o *id* da sua transição original (*Master*). O seu novo conjunto de saída na rede resultante terá incluído apenas o identificador do lugar de corte. O novo conjunto de entrada permanece vazio.

### Sub-Algoritmo 2: Criação dos novos elementos da rede resultantes da aplicação das Regras do Método *Net Splitting*

(Linhas 1 – 32) - Tanto na aplicação da Regra 2 como na aplicação da Regra 3, existe a necessidade de criar um número de cópias das transições de corte igual ao número de sub-redes que apresentam no seu conjunto de elementos de corte que possuem ligações com elementos da sub-rede, menos uma unidade (Pois uma das sub-redes irá conter a transição de corte original).

(Linhas 1 - 16) - Para cada transição de corte identificada em cada posição do Vetor *Cutting_set* com localização definida na mesma posição do Vetor *Master_Loc*, é verificado se o seu identificador se encontra no conjunto de elementos que possuem ligações com elementos da sub-rede de cada estrutura de sub-redes. Caso positivo, se o elemento identificado na mesma posição do Vetor *Master_Loc* tiver o seu identificador no conjnto de elementos da sub-rede, então o identificador da transição de corte no campo *cs_node*s irá manter-se inalterado e é criada uma estrutura *Lts_copy* para a transição de corte original (*Master*). O identificador da transição *Master* é removido do campo *output_nodes* da estrutura da sub-rede visto não ser necessário qualquer cópia de elementos adicionais desta sub-rede, em resultado da aplicação da Regra 3.

***Sub-Algoritmo 2*** *– Criação das cópias das transições de corte (aplicação da Regra 2 e 3)*

Parâmetros Relevantes:
- Vetor de estruturas de elementos da rede inicial, *element - Net_elements*;
- Vetor de estruturas de elementos de corte, *cutting_elem - Cutting_set*;
- Vetor de estruturas de Sub-redes, *subnet – Subnets*;
- Vetor com identificadores de transições de corte – *T_Cutting_set*;
- Vetor de estruturas de conjuntos de transições síncronas e cópias de elementos, *Lts_copy – Lts_copies*;

```
1:  Para cada posição x em T_Cutting_set, tal que Master_Loc[x] ≠ " 0 "  faz {
2:      Para cada posição y em Subnets  faz {
3:          Para n, tal que Subnets[y].cs_nodes[n] = T_cutting_set[x]  faz {
4:              Se ( Existe k, tal que Master_Loc[x] = Subnets[y].nodes[k]  então {
5:                  - Cria estrutura Lts_copy da Transição de Corte(Master);
6:                  Se (Existe h, tal que Subnets[y].output_nodes[h] = T_Cutting_set[x]) então {
7:                      - Remover o valor em Subnets[y].output_nodes[h] e colocar valor nulo;
8:                  }
9:              }
10:             Se (Não Existe k, tal que M_SS_Loc[x] = Subnets[x].nodes[k] ) então {
11:                 - Cria estrutura Lts_copy da Cópia da Transição de Corte(Slave);
12:                 - Subsitui o valor em Subnets[y].cs_nodes[n] pelo id da nova Cópia(Slave) criada;
13:             }
14:         }
15:     }
16: }
17: Para cada posição x em T_Cutting_set, tal que Master_Loc[x] = " 0 "  faz {
18:     Para cada posição y em Subnets  faz {
19:         Para n, tal que Subnets[y].output_nodes[n] = T_cutting_set[x]  faz {
20:             - Cria estrutura Lts_copy da Transição de Corte(Master);
21:             - Remover o valor em Subnets[y].output_nodes[h] e colocar valor nulo;
22:             Para cada posição z em Subnets faz {
23:                 Se (Existe k, tal que Subnets[z].cs_nodes[k] = T_Cutting_set[x] então {
24:                     Se (Subnets[z].id ≠ Subnet[y].id) então {
25:                         - Cria estrutura Lts_copy da Cópia da Transição de Corte(Slave);
26:                         - Subsitui o valor em Subnets[z].cs_nodes[k] pelo id da nova Cópia(Slave) criada;
27:                     }
28:                 }
29:             }
30:         }
31:     }
32: }
```

(Linha 5) - A estrutura *Lts_copy* de cada transição *Master* criada irá ter atribuído o seu identificador único, o seu tipo e o atributo *Master* (*atribute* = 1). É guardado em *copies_id* os identificadores de todas as cópias criadas deste elemento. No seu novo conjunto de entrada e de saída irá ter todos os elementos da rede inicial que após a decomposição pertencem à mesma sub-rede.

(Linha 11) - A estrutura *Lts_copy* de cada transição *Slave* criada irá ter atribuído um novo identificador único para a rede decomposta, o seu tipo e atributo *Slave* (*atribute* = 0). É guardado em *copies_id* o identificador da transição Original. No seu novo conjunto de entrada e de saída irá ter todos os elementos que na rede inicial possuem ligações com a transição *Master* respetiva e pertencem à mesma sub-rede após a decomposição

### Sub-Algoritmo 3: Criação das cópias de elementos da rede resultantes da aplicação da Regra 3

Linhas (1 – 20) - Para finalizar o processo de criação de cópias de elementos, verificou-se se existe identificadores de alguma transição de corte no campo *output_nodes* de cada sub-rede, se existir, é dada indicação de que é necessário criar cópias de elementos da rede inicial (resultantes da aplicação da Regra 3). Nessa mesma sub-rede são criadas as estruturas *Lts_copy* dos lugares contidos no conjunto de entrada dessa transição de corte na rede inicial, e das transições contidas no conjunto de entrada desses lugares. É também criada a estrutura para cada cópia de cada lugar e transição copiado. Para estas últimas transições e para as suas cópias, o processo de criação das suas estruturas *Lts_copy* e dos conjuntos de transições síncronas respetivos é idêntico ao das transições de corte.

(Linha 4) - As estruturas *Lts_copy* dos lugares originais da rede inicial a serem copiadas terão identificador único igual, tipo *place*, e atributo a indicar ser um elemento original da rede inicial. Em *copies_id* é guardado um novo identificador único na rede resultante que será atribuído à cópia. Nestes lugares os novos conjuntos de entrada e de saída mantêm-se vazios.

(Linha 5) -As estruturas *Lts_copy* das cópias dos lugares irão ter associadas um novo identificador único na rede resultante. Em *copies_id* é colocado o identifcador do lugar original respetivo. No seu conjunto de saída irá ter o identificador da transição de corte que na rede inicial tinha ligação com o lugar criado.

(Linha 10) - A estrutura *Lts_copy* de cada transição no conjunto de entrada do lugar copiado terá atribuído o seu identificador único, o mesmo tipo e atributo *Master* (*atribute* = 1). É guardado em *copies_id* o novo identificador único da cópia por criar. Ambos os novos conjuntos de entrada e de saída mantêm-se vazios.

***Sub - Algoritmo 3*** *– Criação das cópias de elementos da rede resultantes da aplicação da Regra 3*

Parâmetros Relevantes:
- Vetor de estruturas de elementos da rede inicial, *element* - Net_elements;
- Vetor de estruturas de Sub-redes, *subnet* – *Subnets*;
- Vetor de estruturas de conjuntos de transições síncronas e cópias de elementos, *Lts_copy* – *Lts_copies*;

```
1:  Para cada  posição x em Subnets  faz {
2:      Para todos  n e y, tal que Subnets[x].output_nodes[n] = Net_elements[y].id   faz {
3:          Para todos  h e i, tal que Net_elements[y].preset[h] = Subnets[x].nodes[i]   faz {
4:              - Cria estrutura Lts_copy do Lugar identificado em Net_elements[y].preset[h];
5:              - Cria estrutura Lts_copy da Cópia do Lugar identificado em Net_elements[y].preset[h];
6:              Para cada z em Net_elements, tal que(Net_elements[z].id = Net_elements[k].preset[h]   faz {
8:                  Para cada j em Net_elements[z].preset[j]   faz {
9:                      Se (Não Existe w, tal que Lts_copy[w].id = Net_elements[z] ) então {
10:                         - Cria estrutura Lts_copy da Transição com id em Net_elements[z].preset[j];
11:                         - Cria estrutura Lts_copy da Cópia da Transição com id em Net_elements[z].preset[j];
12:                     }
12:                     Caso Contrário {
13:                         - Guarda na estrutura Lts_copy já criada informação sobre nova cópia Slave;
14:                         - Cria estrutura Lts_copy da Cópia da Transição com id em Net_elements[z].preset[j];
15:                     }
16:                 }
17:             }
18:         }
19:     }
20: }
```

(Linhas 11 e 14) - A estrutura Lts_copy de cada cópia da transição no conjunto de entrada do lugar copiado terá atribuído um novo identificador único da rede decomposta criado, o mesmo tipo e atributo *Slave* (*atribute* = 1). É guardado em *copies_id* o identificador da transição original copiada. No seu novo conjunto de saída irá ter o identificador da cópia do lugar.

### Criação do Ficheiro PNML da Rede Decomposta:

Este Algoritmo descreve todo o processo de modificação do ficheiro *PNML* com o resultado da decomposição. Informação sobre todos os novos elementos a serem criados e ligações a serem criadas ou reconstruídas encontra-se, nesta fase, presente nas estruturas *Lts_copy*. Este Algoritmo encontra-se dividido em três sub-Algoritmos.

No primeiro sub-Algoritmo é descrito o processo de criação dos canais de comunicação e de arcos que ligam todas as transições de cada conjunto de transições síncronas aos canais respetivos. Ainda no primeiro Algoritmo é descrito o processo de criação das cópias de elementos com

estruturas *Lts_copy* (com valor *atribute = 0*) criadas, e a eliminação dos nós dos arcos no ficheiro *PNML* que ligam elementos do conjunto de corte a outros elementos da rede inicial.

O sub-Algoritmo 2 descreve o processo de reconstrução de arcos entre os elementos de corte originais com estrutura *Lts_copy* e elementos da sub-rede onde estes se localizam após a decomposição.

O sub-Algoritmo 3 descreve o processo de criação de novos arcos que ligam as cópias de elementos da rede inicial (com estrutura *Lts_copy* criada e atributo *Slave*) com elementos da sub-rede onde se localizam após a decomposição.

### Sub-Algoritmo 1: Criação do Ficheiro PNML da Rede Decomposta

(Linhas 1 – 9) – Procura-se no ficheiro *PNML*, no interior do nó da rede, por todos os arcos que possuem no identificador de elemento de origem (*source*) ou de incidência (*target*), um identificador igual ao de um elemento de corte seleccionado. Caso positivo, o nó do arco é eliminado.

(Linhas 10 – 21) – Para cada elemento da rede inicial com estrutura *Lts_copy* criada (elemento original, *atribute* = 1) é clonado o nó do ficheiro *PNML* respetivo, sendo este alterado de modo a remover-se todas as guardas de sinais e eventos e a substituir-se o identificador do nó clonado com os identificadores das cópias deste elemento (*copies_id*). É inserido na rede um clone modificado para cada cópia deste elemento da rede.

(Linhas 22 – 30) - Para cada estrutura *Lts_copy* de transições *Master*, é criado um canal de comunicação. O tipo do canal criado depende da especificação deste por parte do utilizador no editor. Este poderá ser síncrono ou assíncrono. É também criado um arco que liga a transição *Master* ao canal e vários arcos que ligam o canal às transições *Slaves*.

***Sub - Algoritmo 1 -*** *Criação do Ficheiro PNML da Rede Decomposta*

Parâmetros Relevantes:
- Vetor de estruturas de elementos da rede inicial, *element - Net_elements;*
- Vetor de estruturas de conjuntos de transições síncronas e cópias de elementos, *Lts_copy – Lts_copies*;

```
1:  Para cada Nó ELEM no Nó da rede(PNML) faz {
2:      Se (Nó ELEM é um arco) então {
3:          Para cada posição x em Cutting_set faz {
4:              Se (Origem do Nó ELEM = Cutting_set[x].id) então {
5:                  - Elimina Nó ELEM do Nó da Rede (Elimina Arco);
6:              }
7:              Se (Destino do Nó ELEM = Cutting_set[x].id) então {
8:                  - Elimina Nó ELEM do Nó da Rede (Elimina Arco);
9:              }
7:          }
8:      }
9:  }
10: Para cada Nó ELEM no Nó da Rede (PNML) faz {
11:     Para cada posição x em Lts_copies faz {
12:         Se Lts_copies[x].id = id do Nó ELEM e Lts_copies[].atribute = 1 então {
13:             Para cada n em Lts_copies[x].copies_id[n], tal que Lts_copies[x].copies_id[n] ≠  faz {
14:                 - Clona Nó ELEM para um Nó ELEM_aux (PNML);
15:                 - Altera id do Nó ELEM_aux com o valor do id em Lts_copies[x].copies_id[n](PNML);
16:                 - Remove Guardas de Sinais e Eventos do Nó ELEM_aux(PNML);
17:                 - Cria elemento ELEM_aux (PNML);
18:             }
19:         }
20:     }
21: }
22: Para cada posição x em Lts_copies faz {
23:     Se Lts_copies[x].type = transição e Lts_copies[x].atribute = 1 então {
24:         - Cria Canal no nó da rede do Ficheiro PNML;
25:         - Cria Arco no Ficheiro PNML com origem Lts_copies[x].id  e destino Canal Criado;
26:         Para cada n em Lts_copies[x].copies_id[n], tal que Lts_copies[x].copies_id[n] ≠ nulo faz {
27:             - Cria Arco (PNML): Origem Canal Criado, Destino Lts_copies[x].copies_id[n];
28:         }
29:     }
30: }
```

### Sub-Algoritmo 2: Criação do Ficheiro PNML da Rede Decomposta

São criados, no nó da rede do ficheiro *PNML*, arcos a ligarem os elementos originais da rede inicial, com estrutura *Lts_copy* criada, com os elementos identificados no seu novo conjunto de saída (*New_postset).*

(Linhas 12 – 21) – São criados, no nó da rede do ficheiro *PNML*, arcos a ligarem os elementos originais da rede inicial, com estrutura *Lts_copy* criada, com os elementos identificados no seu novo conjunto de entrada (*New_preset*).

***Sub - Algoritmo 2** - Criação do Ficheiro PNML da Rede Decomposta*

Parâmetros Relevantes:
- Vetor de estruturas de elementos da rede inicial, *element - Net_elements;*
- Vetor de estruturas de conjuntos de transições síncronas e cópias de elementos, *Lts_copy – Lts_copies;*

```
1:  Para cada posição x em Lts_copies faz {
2:      Se (Lts_copies[x].atribute = 1 (Original/Master) então{
3:          Para cada n em Lts_copies[x].New_postset[n] faz {
4:              Para cada posição y em Net_elements, tal que Net_elements[y].type = arco faz {
5:                  Se (Net_elements[y].preset[0] = Lts_copies[x].id) então {
6:                      Se (Net_elements[y].postset[0] = Lts_copies[x].New_postset[n]) então {
7:                          - Cria Arco (PNML): Origem Lts_copies[x].id, Destino Lts_copies[x].New_postset[n]);
8:                      }
9:                  }
10:             }
11:         }
12:         Paea cada n em Lts_copies[x].New_preset[n] faz {
13:             Para cada posição y em Net_elements, tal que Net_elements[y].type = arco faz {
14:                 Se (Net_elements[y].preset[0] = Lts_copies[x].New_preset[n]) então {
15:                     Se (Net_elements[y].postset[0] = Lts_copies[x].id então {
16:                         - Cria Arco (PNML): Origem Lts_copies[x].New_preset[n]), Destino Lts_copies[x].id;
15:                     }
17:                 }
18:             }
19:         }
20:     }
21: }
```

### Sub-Algoritmo 3: Criaçãodo Ficheiro PNML da Rede Decomposta

(Linhas 1 – 11) – São criados, no nó da rede, do ficheiro *PNML*, arcos a ligarem os elementos originais da rede inicial, com estrutura *Lts_copy* criada, com os elementos identificados no seu novo conjunto de saída (*New_postset*).

(Linhas 12 – 21) – São criados, no nó da rede, do ficheiro *PNML*, arcos a ligarem as cópias dos elementos da rede inicial, com estrutura *Lts_copy* criada, com os elementos identificados no seu novo conjunto de entrada (*New_preset*).

***Sub - Algoritmo 3**: Criação do Ficheiro PNML da Rede Decomposta*

Parâmetros Relevantes:
- Vetor de estruturas de elementos da rede inicial, *element - Net_elements*;
- Vetor de estruturas de conjuntos de transições síncronas e cópias de elementos, *Lts_copy – Lts_copies*;

```
1:  Para cada posição x em Lts_copies faz {
2:      Se Lts_copies[x].atribute = 0 (Cópia/Slave) então{
3:          Para cada n em Lts_copies[x].New_postset[n] faz {
4:              Para cada posição y em Net_elements, tal que Net_elements[y].type = arco faz {
5:                  Se (Net_elements[y].preset[0] = Lts_copies[x].copies_id[0]) então {
6:                      Se(Net_elements[y].postset[0] = Lts_copies[x].New_postset[n]) então {
7:                          - Cria Arco (PNML): Origem Lts_copies[x].id, Destino Lts_copies[x].New_postset[n]);
8:                      }
9:                  }
10:             }
11:         }
12:         Para cada n em Lts_copies[x].New_preset[n] faz {
13:             Para cada posição y em Net_elements, tal que Net_elements[y].type = arco faz {
14:                 Se (Net_elements[y].preset[0] = Lts_copies[x].New_preset[n]) então {
15:                     Se (Net_elements[y].postset[0] = Lts_copies[x].copies_id[0]) então {
16:                         - Cria Arco (PNML): Origem Lts_copies[x].id, Destino Lts_copies[x].New_preset[n];
15:                     }
17:                 }
18:             }
19:         }
20:     }
21: }
```

## 4.4 Instruções de utilização da ferramenta *SPLIT*

A ferramenta implementada pode ser utilizada a partir do editor da plataforma *IOPT*-Tools[20]. O texto que se segue oferece indicações para como utilizar a ferramenta *SPLIT*.

Antes da evocação do *plug-in SPLIT* é necessário definir-se primeiro os parâmetro relevantes à decomposição, tais como: os elementos de corte; pares de elementos da mesma sub-rede; e localização da transição Master de transições de corte com lugares de entrada pertencentes a diferentes sub-redes;

1 - Seleccionar no editor, um conjunto de nós onde se pretende realizar a decomposição da rede *IOPT*. (Nota: Não devem existir ligações entre os nós de corte seleccionados. Removendo estes nós da rede *IOPT*, o número de redes isoladas resultante deve ser pelo menos dois).

2 - No campo *Comment* (Figura 4.8) do menu de propriedades de elementos da rede (no editor da plataforma *IOPT*-Tools) de transições de corte, caso os seus lugares de entrada na rede inicial pertençam a diferentes sub-redes após a decomposição, preencher com o identificador do elemento que pertence à sub-rede onde se pretende que a transição de corte Master se localize.(OPCIONAL).

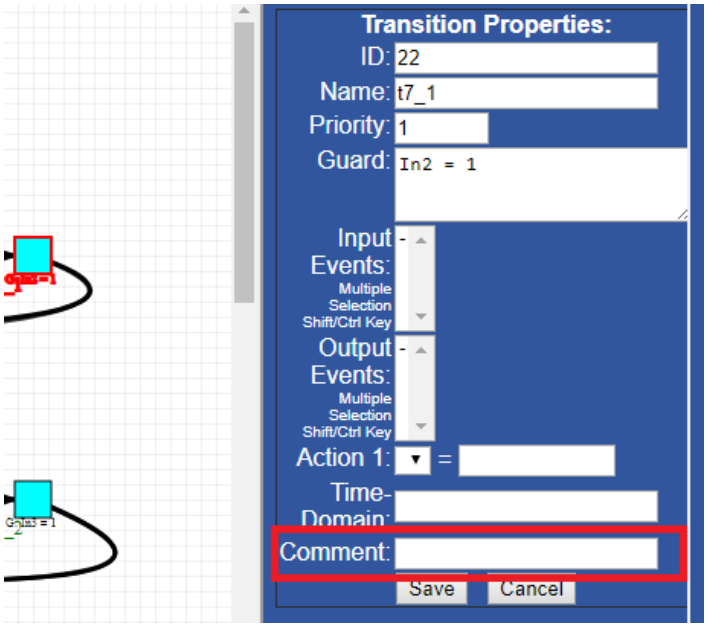


Figura 4.8 – Campo Comment nas propriedades dos nós da rede no editor da plataforma *IOPT-Tools*.

3 - No mesmo campo *Comment* do menu de propriedades de elementos da rede que não pertencem ao conjunto de corte seleccionado, preencher com o identificador de outro elemento da rede que se pretende que após a decomposição se localize na mesma sub-rede (OPCIONAL).

4- Após definidos os parâmetros e seleccionados os elementos de corte procede-se à evocação do *plug-in SPLIT* que se encontra na barra de plug-ins do menu de edição da *IOPT-Tools* (*Figura 4.9*).

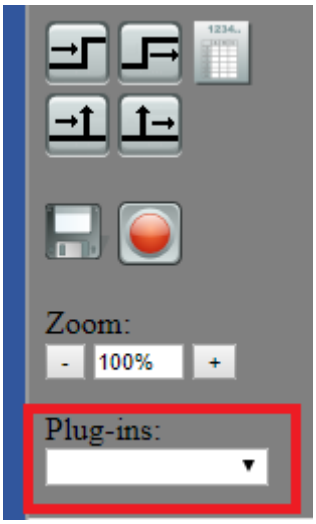


Figura 4.9 – Barra de Plug-in's do editor da plataforma *IOPT*-Tools.

5 - Evocado o *plug-in* (Formulário da *Figura 4.10*) procede-se à definição do tipo de canais de comunicação (Síncronos ou Assíncronos) entre sub-redes, no campo *ImpType*. Premir *Apply* para executar a decomposição.

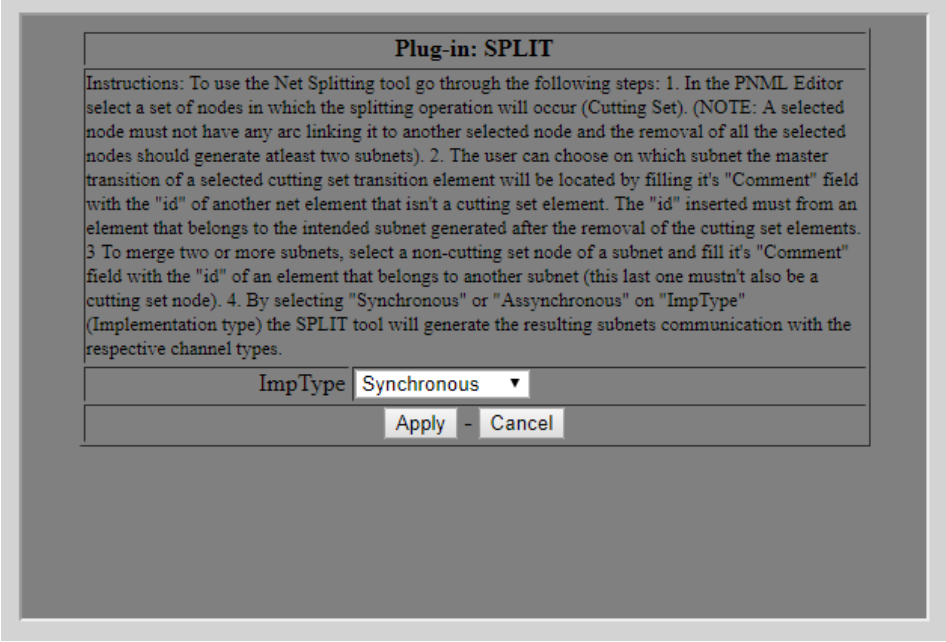


Figura 4.10 – Formulário evocado do Plug-in *SPLIT* no editor da plataforma *IOPT*-Tools.

# Capítulo 5 - Casos de Aplicação

Neste Capítulo são apresentado dois exemplos onde foi aplicada a ferramenta de *SPLIT* e é, consequentemente, divulgada a sua capacidade e eficácia na preparação dos controladores individuais obtidos.

O primeiro exemplo descreve a interacção entre um emissor e um receptor, sendo que o seu objetivo é demonstrar a aplicabilidade das regras #1 e #2 do método no qual se baseou a implementação da ferramenta, ou seja, para testar o resultado nos casos em que os elementos de corte seleccionados são lugares e transições que possuem lugares de entrada pertencentes à mesma sub-rede. O segundo exemplo representa o controlador de um parque de estacionamento com duas entradas e uma saída, o objetivo deste é observar o resultado da aplicação da terceira regra do método já referido, ou seja, quando os elementos de corte seleccionados são transições com elementos de entrada pertencentes a diferentes sub-redes.

## 5.1 Emissor - Receptor

De modo a simplificar o estudo, utilizou-se o simples exemplo que representa a interacção de um receptor com um emissor na *Figura 5.1.* Num estado inicial, a rede da *Figura 5.1* apresenta uma única marcação no lugar *Msg_Sready* e no lugar *Msg_Rready* que representa o estado em que o emissor se encontra preparado para enviar uma mensagem ao receptor e o estado em que o receptor se encontra preparado para receber uma mensagem do emissor, respetivamente. Quando o emissor envia a mensagem ao receptor (disparo da transição *Send_msg*), este aguarda o envio de uma mensagem por parte do receptor ("*Acknowlege*") representado pelo lugar *Ack_wait*, dando informação ao emissor de que a mensagem enviada foi recebida com sucesso e que o canal se encontra livre para que o emissor envie uma nova mensagem. Na recepção da mensagem a transição *Receive_msg* dispara apenas caso o receptor esteja preparado para receber uma mensagem e exista uma mensagem enviada (marcação no lugar Message). Caso positivo, o receptor recebe a mensagem colocando uma marcação no lugar *Msg_received*, indicando que a mensagem foi recebida com sucesso.

.

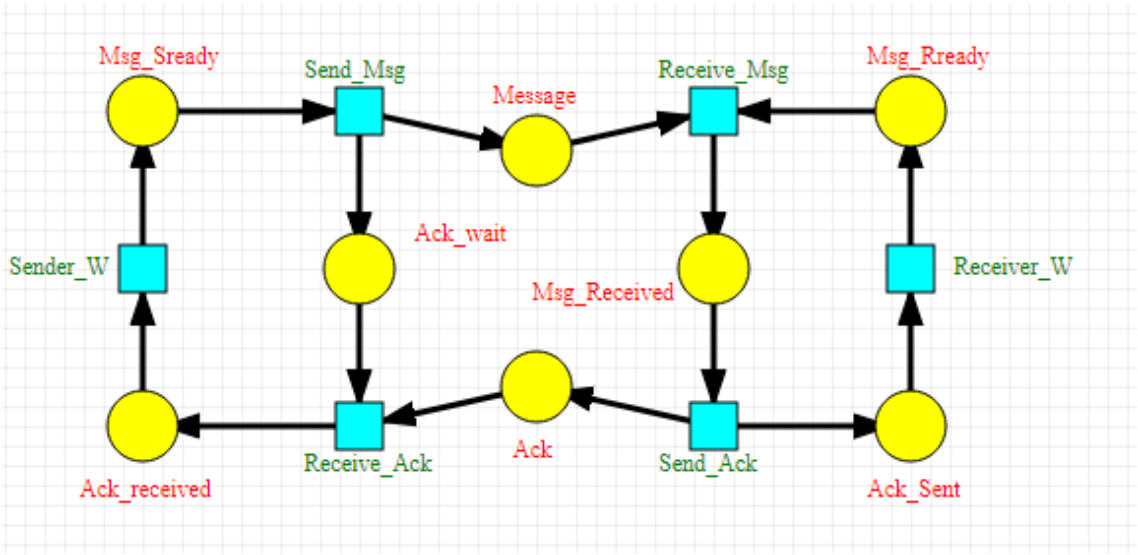


Figura 5.1 – Rede de Petri descritiva de um sistema emissor-receptor.

Neste modelo consegue-se identificar os dois componentes distintos deste sistema, emissor e receptor. Assumindo que pretendemos representar o controlador de cada um destes componentes de forma independente mas mantendo o comportamento do modelo inicial (descrevendo comunicação por meio de canais síncronos direccionais entre os componentes) procedemos à aplicação da ferramenta *SPLIT* desenvolvida.

Começamos por identificar os elementos onde podemos efectuar a decomposição pretendida. Uma sub-rede com a descrição do comportamento do emissor juntamente com o processo de envio de mensagem e outra para o comportamento do receptor e o processo de envio da mensagem de confirmação. Para este fim, identificamos quatro combinações possíveis de pares de elementos que podem ser escolhidos como elementos do conjunto de corte definido.

Escolhendo os lugares *Ack* e *Message* estamos perante a aplicação da primeira regra do método *Net Splitting*, em que a implementação da ferramenta *SPLIT* se baseou, para ambos os elementos seleccionados. Na *Figura 5.2* é ilustrado o resultado da remoção de ambos os lugares seleccionados da rede. Ambos os lugares possuem uma só transição de entrada. Na aplicação da regra #1 os lugares seleccionados irão ser adicionados ao componente onde se encontra a transição dos seus conjuntos de saída e ligados a esses mesmos elementos. As transições do conjunto de entrada dos lugares *Message* e *Ack*, são copiadas, é atribuída a etiqueta master às transições originais, sendo estas posteriormente adicionadas ao componente que não possui o lugar do conjunto de corte ao qual se encontravam ligadas no modelo inicial. A cópia das transições com atributo *Slave* é adicionada ao componente onde se encontra o lugar de corte seleccionado respetivo.

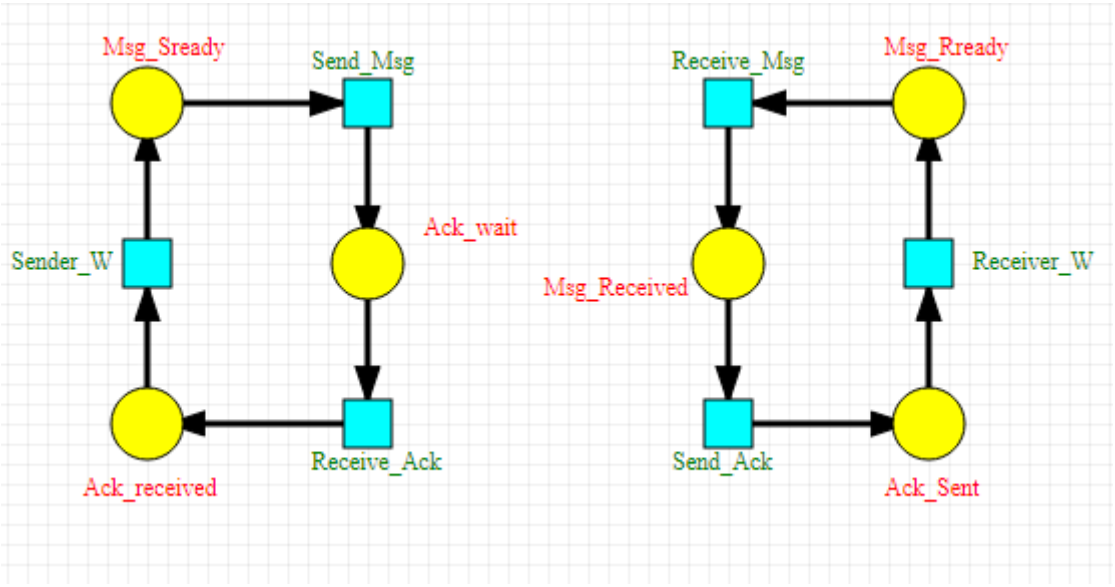


Figura 5.2 – Resultado da remoção dos lugares Message e Ack do modelo inicial.

No entanto, escolhendo as transições *Send_msg* e *Send_ack* estariamos perante a aplicação da segunda regra, em que em ambas as transições os seus lugares de entrada pertencem a uma única sub-rede. *A Figura 5.3* representa o resultado da remoção das transições *Send_Msg* e *Send_Ack* da rede inicial. Na aplicação desta segunda regra, as transições seleccionadas como elementos do conjunto de corte, *Send_msg* e *Send_ack* irão ser copiadas, sendo que, a transição original com atributo *Master* irá ser adicionada ao compoente que possui os elementos do seu conjunto de entrada e a transição com atributo *Slave* ao componente que possui os elementos do seu conjunto de saída.

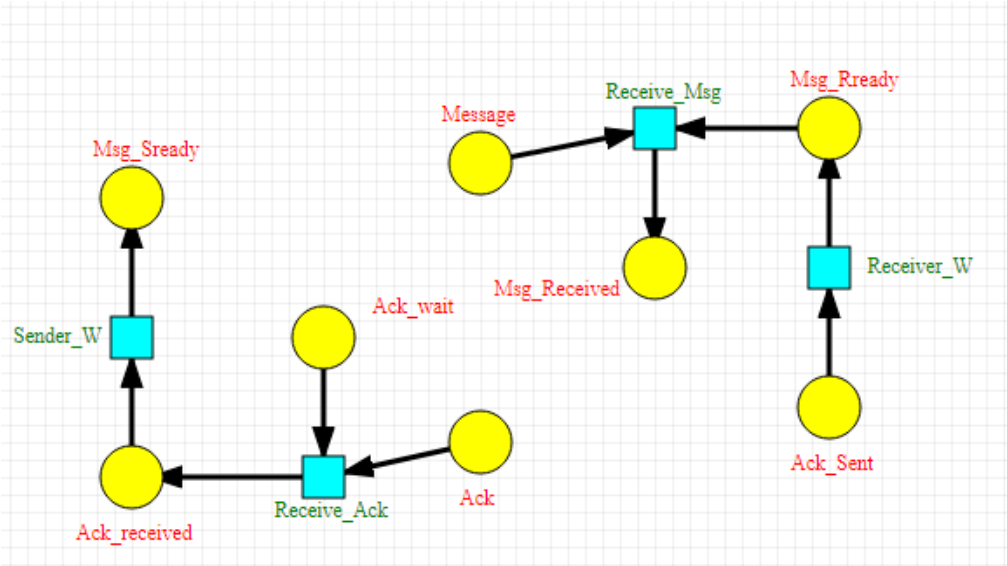


Figura 5.3 – Resultado da remoção das transições Send_msg e Send_ack da modelo inicial.

Tal como descrito, o resultado gerado para ambos os conjuntos de corte definidos deve ser idêntido, e tal é verificado com a aplicação da ferramenta *SPLIT*. Após a aplicação da ferramenta de decomposição em ambos os casos o resultado é o ilustrado na *Figura 5.4*.

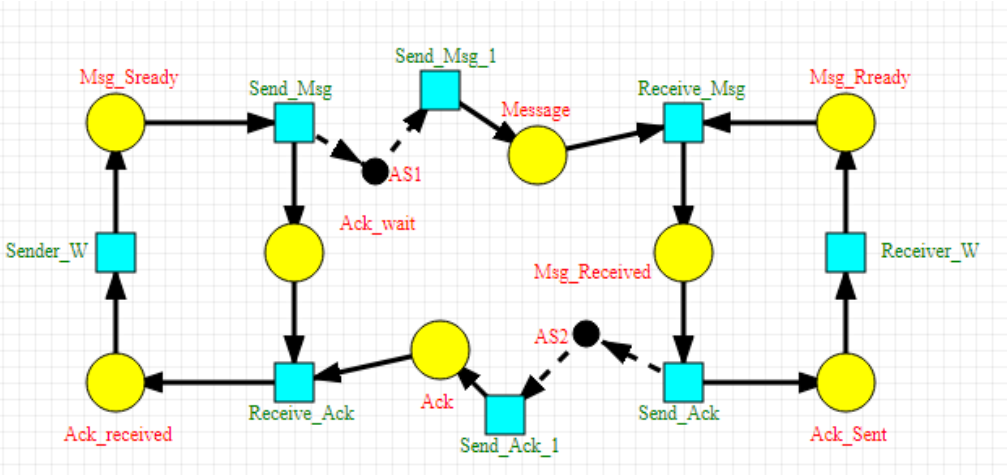


Figura 5.4 – Resultado da Aplicação da ferramenta *SPLIT* no modelo emissor- receptor para ambos os conjuntos de corte definidos (Primeiro conjunto: lugares *Message e Ack*; Segundo conjunto: transições *Send_msg e Send_Ack*).

É de se salientar a aplicabilidade desta ferramenta não só para a classe *IOPT* de redes de Petri mas também para outras classes. No caso da aplicação nas redes *IOPT*, a transição com atributo *Master* permanecerá com todos os sinais de entrada que a transição original possuía e todas as transições *Slaves* (cópias) a estas ligadas por meio de canais síncronos ou assíncronos encontram-se desprovidas destes sinais, sendo que o seu disparo depende só do sinal gerado pelo disparo da transição *Master*.

## 5.2 Parque de Estacionamento

O seguinte modelo representado na *Figura 5.5* descreve o controlador de um parque de estacionamento. Várias configurações do parque de estacionamento podem ser assumidas dependendo do número de entradas, saídas e andares que o utilizador queira inserir no seu modelo, no entanto, neste exemplo consideramos um parque de estacionamento com duas entradas, uma saída e um andar.

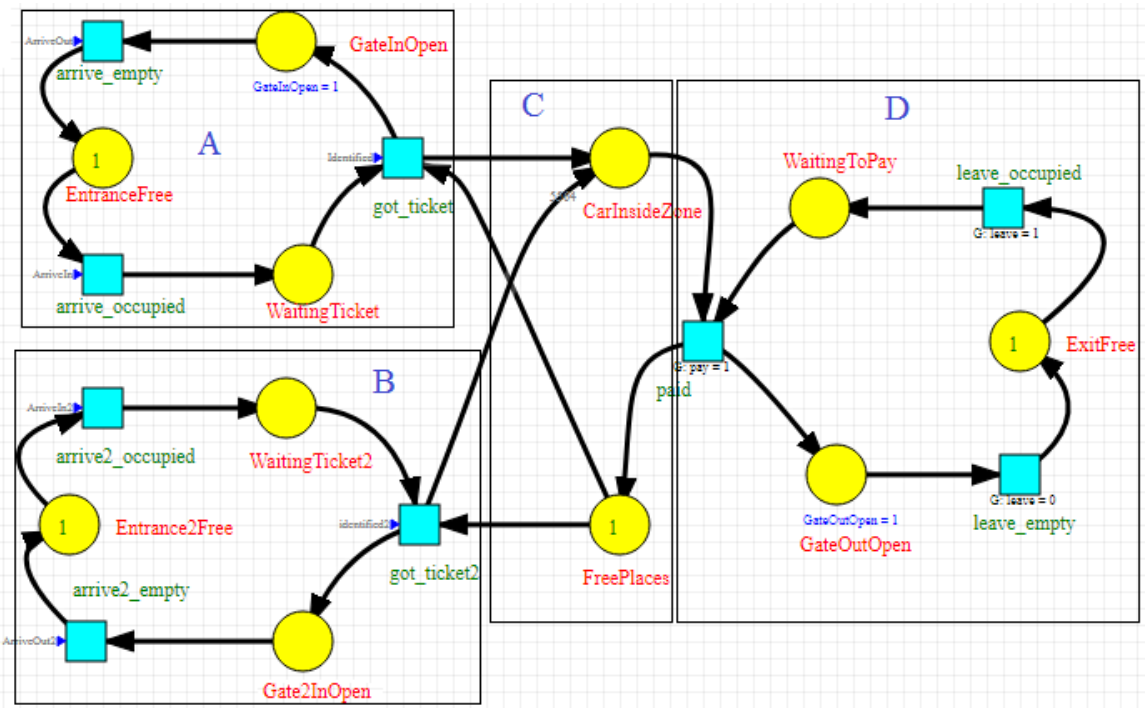


Figura 5.5 – Rede de Petri *IOPT* do sistema de um parque de estacionamento com 2 entradas e 1 saída.

As características do sistema pretendido e descrito na *Figura 5.5* são as seguintes:

- Duas entradas – O processo de entrada de carros é descrito pelos troços A e B , que correspondem à primeira e segunda entrada, respetivamente;

- Uma saída – Controlador da única saída de carros, é descrito pelo troço D;

- Zona de estacionamento – Constituída por dois lugares com informação sobre o número de lugares ocupados e livres do parque (Troço C);

Durante a descrição do sistema, é dado o mesmo nome dos sinais às transições nas quais estes se encontram atribuídos. Duas cancelas bloqueiam as portas de ambas as entradas. Num estado inicial ambas as cancelas encontram-se fechadas, e o modelo é inicializado com uma marcação no lugar *EntranceFree*, indicando que a entrada se encontra livre. Com a chegada de um carro a uma das entradas, um sensor de presença activa o sinal *arrive_occupied*, mudando o estado do sistema para *WaitingTicket* (Aguardando bilhete). Para ser possível a entrada de um carro no parque de estacionamento, é necessário que haja lugares livres no interior do parque e que o condutor da viatura peça o bilhete de entrada para que as cancelas abram. Quando o condutor pede o bilhete, o sinal *got_ticket* é activado, gerando um evento responsável pela abertura das cancelas. Após a entrada do carro no parque de estacionamento, uma marcação nos lugares livres é decrementada e uma nos lugares ocupados é incrementada (identificados pelos lugares

*CarsInsideZone* que representa os lugares ocupados, e *FreePlace*s que tal como o nome indica representa os lugares livres).

Como observado, os controladores de ambas as entradas são semelhantes, esta característica do desenvolvimento baseado em modelos trata-se de uma enorme vantagem, pois permite uma fácil reutilização de sub-modelos já identificados e desenvolvidos para aumentar a complexidade do sistema em questão, como por exemplo, facilmente adicionar um maior número de entradas e saídas no caso do modelo do parque de estacionamento.

Na saída do parque de estacionamento, tal como na entrada, uma cancela encontra-se fechada. Este troço de rede também se encontra inicializado com uma marcação no lugar *ExitFree* que indica a ausência de qualquer carro na saída. Após a chegada de um carro à zona da saída, um sensor de presença activa o sinal leave_occupied mudando o estado do sistema para *WaitingToPay*, que representa o aguardar do pagamento pelo condutor para abrir a cancela e permitir a saída da viatura. Após o condutor efectuar o pagamento o sinal *paid* é activado, a cancela de saída abre e a viatura é livre de abandonar o local.

O exemplo do parque de estacionamento com múltiplas entradas e/ou saídas, tal como o exemplo das duas vagonetes mencionado anteriormente, é um bom exemplo de interesse para aplicação da ferramenta *SPLIT*. Neste caso, é possível indentificar-se na composição do modelo global, vários sub-modelos que representam a descrição do comportamente de uma unidade independente (componente) do sistema (entradas, saídas, e lugares livres e ocupados dentro do parque de estacionamento). Assumindo que se queira dividir o sistema global para uma posterior implementação do controlador de cada um dos sub-modelos identificados em plataformas distintas, mantendo o comportamento do sistema inicialmente descrito, começamos por identificar os elementos da rede cuja sua remoção resulta em múltiplas sub-redes.

Seleccionando apenas as transições *got_ticket*, *got_ticket2* e *paid* como elementos do conjunto de corte e removendo-as da rede, o resultado é o ilustrado na Figura 5.6. Como nenhum elemento do conjunto de corte possui ligação directa com outro elemento do mesmo conjunto, a primeira etapa de validação encontra-se concluída. É também relevante mencionar-se, neste exemplo, que as transições *got_ticket* e *got_ticket2* encontram-se numa situação de conflito, o que significa que , segundo o método *Net Splitting*, é obrigatório que ambas as transições com atributo *Master*, após a decomposição, permaneçam na mesma sub-rede para garantir manter o comportamento do sistema inicial no caso de se adoptar uma execução distribuída dos diversos componentes. Todas as transições seleccionadas como elementos do conjunto de corte possuem lugares de entrada pertencentes a mais do que uma sub-rede, sendo a terceira regra do método a aplicada neste exemplo.

Na *Figura 5.6* observa-se que o resultado da remoção das transições seleccionadas resulta em cinco sub-redes isoladas e, consequentemente, no controlador de cinco componentes, um para cada entrada, um para a saída, e dois elementos isolados *FreePlaces* e *CarInsideZone*.

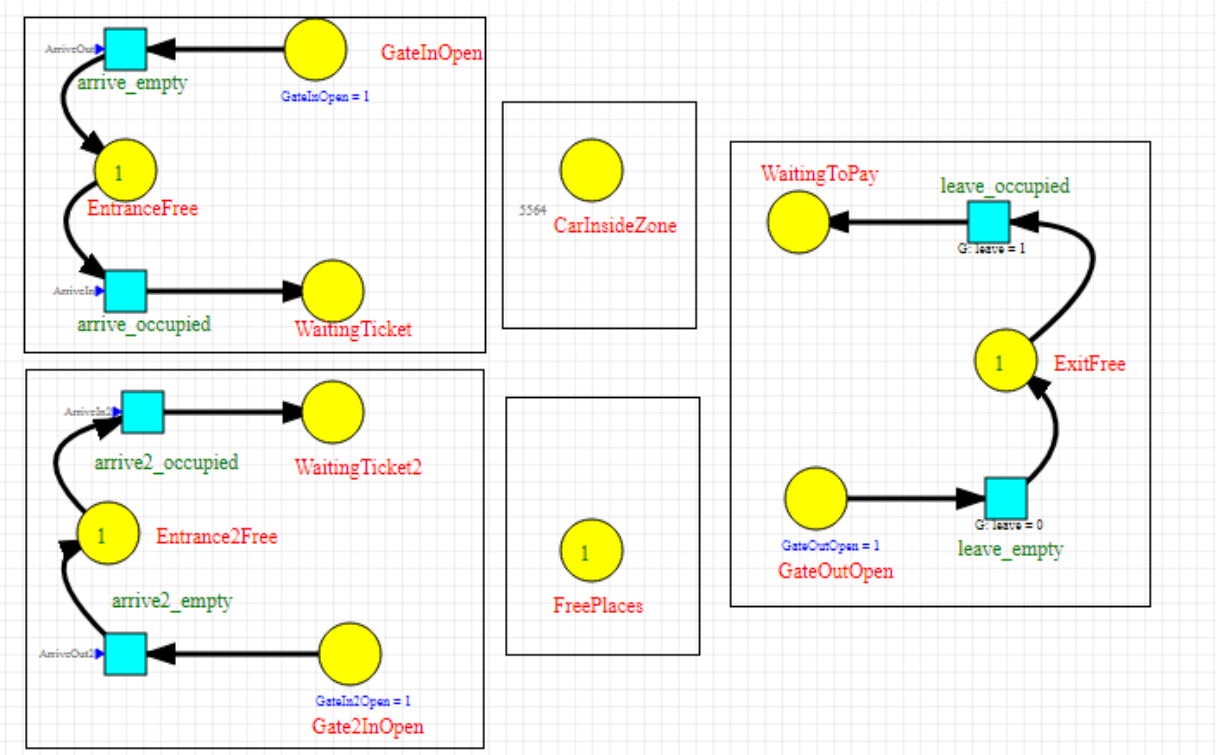


Figura 5.6 – Resultado da remoção das transições *got_ticket, got_ticket2, paid* do modelo do parque de estacionamento.

No entanto, o que se pretende é o resultado ilustrado na *Figura 5.7* que ilustra o resultado da aplicação da ferramenta *SPLIT* implementada no modelo representado na *Figura 5.5*, em que ambos os elementos isolados permanecem na mesma sub-rede após a decomposição, representando assim, um único controlador do componente do interior do parque. Para isto, é necessário para além de se indicar apenas o conjunto de corte, definir-se os elementos que necessitam manter-se na mesma sub-rede No resultado obtido na *Figura 5.7*, foi colocado, no campo *Comment* do lugar *FreePlaces*, o identificador único na rede do lugar *CarInsideZone* de modo a manterem-se na mesma sub-rede.

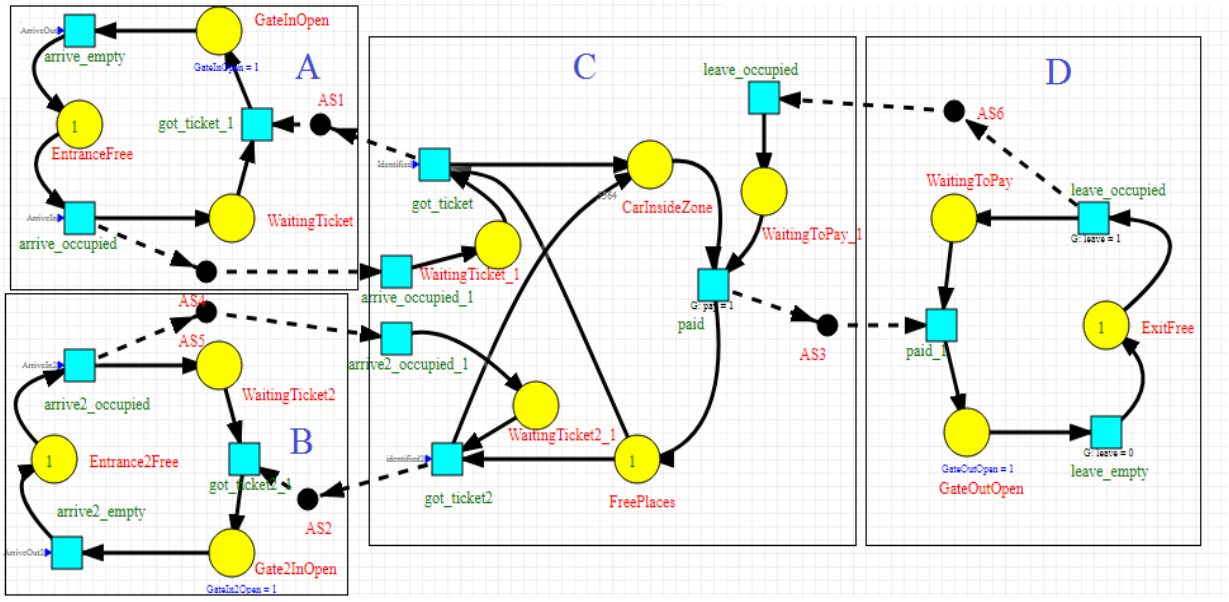


Figura 5.7 – Resultado obtido da aplicação da ferramenta *SPLIT* da plataforma *IOPT-Tools* no modelo do Parque de estacionamento.

Nesta última Figura mencionada observamos que a colocação da transição *paid* com atributo *Master* localiza-se na sub-rede que representa a zona do parque de estacionamento, tal como as outras transições de corte. Visto que esta transição não se encontra numa situação de conflito, ao contrário das outras *(got_ticket, got_ticket2*), é indiferente em qual das sub-redes se poderá incluir a transição Master. Para se alterar a localização da transição *Master* basta incluir na transição *paid* da rede inicial, no seu campo *Comment*, o identificador de um elemento que pertença à sub-rede alvo da sua localização (elemento do troço C ou do Troço D da *Figura 5.5*).

Com a aplicação da terceira regra do método, para garantir o comportamento inicial do sistema foram criadas as cópias dos lugares e transições que antecedem uma transição do conjunto de corte e que pertencem a uma sub-rede cuja transição do conjunto de corte tem o atributo *Slave*. Estas cópias são então ligadas à transição *Master*, que se encontra numa sub-rede diferente, de modo a oferecer informação sobre o estado presente da sub-rede com transição *Slave* à sub-rede com transição *Master*.

# Capítulo 6 - Conclusões e Trabalhos Futuros

## 6.1 Conclusões

No decorrer desta dissertação foi desenvolvido o protótipo da ferramenta *SPLIT*, implementada em linguagem de programação C capaz de efectuar uma decomposição automática (com base no método *Net Splitting*) de um modelo de uma rede de Petri *IOPT* em diversas sub-redes de acordo com determinados parâmetros definidos pelo utilizador. A necessidade desta ferramenta surge na medida que, a utilização de meios manuais para identificação e decomposição do controlador de sistemas digitais (embutidos, de automação, etc…) de elevada complexidade, descrito por uma rede de Petri *IOPT*, em diversos componentes separados, torna-se um processo lento e contra-produtivo.

A ferramenta implementada encontra-se integrada na plataforma *IOPT-Tools [20]*. Embora a ferramenta *SPLIT* tenha sido implementada tomando como referência a classe *IOPT* e procure oferecer suporte ao desenvolvimento de sistemas embutidos e de automação, a sua utilização pode ser extendida a outras plataformas de desenvolvimento de controladores que recorrem ao uso do formalismo de redes de Petri ou a qualquer outra plataforma de edição de modelos em redes de Petri de uma outra classe de redes. No entanto, essas mesmas plataformas devem ser capazes de suportar a utilização de canais síncronos como meio de comunicação entre as transições com atributo *Master* e as transições com atributo *Slave* do mesmo conjunto de transições síncronas, como definidos na plataforma *IOPT-Tools*, e no método *Net Splitting*.

Como observado nos modelos decompostos gerados pela ferramenta *SPLIT* e estudados no capítudo 5, a implementação foi bem sucedida estando a ferramenta pronta para utilização. Os resultados em todos os casos de teste efectuados coincidem com os resultados esperados da aplicação do método *Net Splitting*. No entanto, uma contínua monitorização da utilização desta ferramenta é recomendada de modo a ser possível identificar e corrigir possíveis erros menores de *Software*.

A ferramenta de decomposição oferece ainda a possibilidade de escolher o tipo de comunicação entre os componentes, podendo esta ser síncrona ou assíncrona. A inclusão de canais assíncronos além dos canais síncronos definidos no método operação *Net Splitting*, visa preparar o modelo gerado para uma eventual adopção de uma arquitetura *GALS*, ficando apenas por definir os domínios de tempo de execução de cada componente.

## 6.2 Trabalhos Futuros

Para um demepenho melhorado, recomenda-se a optimização do código C desenvolvido na implementação da ferramenta *SPLIT* para efeitos de melhoria da performance da utilização da ferramenta.

Uma sugestão de projecto futuro passa por estabelecer padrões de compatibilidade entre os ficheiros dos modelos *PNML* gerados por diferentes plataformas de edição de modo a aumentar os níveis de compatibilidade entre as várias plataformas existentes, e consequentemente, expandir a utilidade da ferramenta *SPLIT*, para outras aplicações. Poderá ser feito, para esta finalidade, uma colecção de informação dos ficheiros *PNML* gerados por várias plataformas de desenvolvimento de redes de Petri presentemente existentes, sendo estudadas as diferenças que estas apresentam. Após identificadas as diferenças entre os ficheiros gerados e as diferenças entre as classes de redes de Petri que são tomadas como referência pelas plataformas de edição, poderá ser implementada uma ferramenta de conversão de modelos de uma classe para outra classe, sendo removidos os atributos incompatíveis da classe origem e adicionados os atributos necessários da classe alvo do ficheiro *PNML* respetivo, ou ,conversão de um modelo de uma plataforma para outra que não apresenta compatibilidade com o modelo alvo, sendo feita a remoção de elementos incompatíveis e a sua substituição por elementos compatíveis equivalentes.

# *Referências Bibliográficas*

[1] - Anikó Costa. *Petri Net Model Decomposition – A Model Based Approach Supporting Distributed Execution -. PhD thesis, Universidade Nova de Lisboa, 2010.*

[2] – OMG. UML Resource Page, 2008. http://www.uml.org. *acedido pela última vez em: 19/09/2018*;

[3] - T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE, vol 77, No. 04, April 1989*, pages 541{580, 1989.

[4] - Wolfgang Reisig. *Elements of distributed algorithms: modeling and analysis with Petri nets*. Springer-Verlag New York, Inc., New York, NY, USA, 1998.

[5] - Wolfgang Reisig. *Petri nets: an Introduction*. Springer-VerlagNew York, Inc., 1985.

[6] - Luís Gomes. *Redes de Petri Reactivas e Hierárquicas - Integração de formalismos no projecto de sistemas reactivos de tempo real* -. PhD thesis, Universidade Nova de Lisboa, 1997.

[7] - G. Rozenberg, P. S. Thiagarajan; 1986; "*Petri nets: basic notions, structure, behaviour"; in Current Trends in Concurrency*, LNCS 224; J.W. de Bakker, W. -P. De Roever, G. Rozenberg (Eds.); Springer-Verlag

[8] - JÄorg Desel and Javier Esparza. *Free Choice Petri Nets*. Cambridge University Press, 1995.

[9] - K. Jensen and L.M. Kristensen. *Coloured Petri Nets*. Springer-Verlag, Berlin Heidelberg, 2009.

[10] - Wolfgang Reisig; 1992; "A Primer in Petri Design"; Springer-Verlag; ISBN 3-540-52044-9

[11] - Charles Lakos. The Consistent Use of Names and Polymorphism in the Definition of Object Petri Nets. In *Proceedings of the 17th International Conference on Application and Theory of Petri Nets*, pages 380{399, London, UK, 1996. Springer-Verlag.

[12] - H. J. Genrich, Predicate/Transition Nets, LNCS vol. 254, Springer Verlag, 1987

[13] - René David. Modeling of dynamic systems by Petri nets. In *Proceedings of the ECC91 European Control Conference*, pages 136{147, Grenoble, France, July 2-5 1991.

[14] - M. Silva; 1985; "Las Redes de Petri: en la Automática y la Informática"; Editorial AC, Madrid, ISBN 84 7288 045 1.

[15]- M. Hack; December 1975; “Decidability Questions for Petri Nets”; Ph. D. Dissertation, Dep. Of Electrical Engineering, Massachusets Institute of Technology

[16] - M. Ajmone Marsan. Stochastic Petri nets: An elementary introduction. In *In Advances in Petri Nets*, pages 1{29. Springer, 1989.

[17] - Luís Gomes, João Paulo Barros, Anikó Costa, and Ricardo Nunes. The Input-Output Place/Transition Petri Net Class and Associated Tools. In *5th IEEE International Conference on Industrial Informatics (INDIN 2007)*, Jul. 2007.

[18] - Rui Pais, João Paulo Barros, and Luís Gomes. A Tool for Tailored Code Generation from Petri Net Models. In *10th IEEE Conference on Emerging Technologies and Factory Automation (ETFA 2005).*, Sep. 2005.

[19] - Fernando Pereira, Filipe Moutinho, Luis Gomes. *IOPT-TOOLS* - User Manual. Universidade Nova de Lisboa, Faculdade de Ciências e Tecnologias, GRES Research Group, versão 1.1, 2014 em: http://gres.uninova.pt/iopt_usermanual.pdf.

[20] – IOPT-Tools Webpage: *http://gres.uninova.pt/IOPT-Tools acedido pela última vez em: 19/09/2018*;

[21] - Luis Gomes, Filipe Moutinho, Fernando Pereira; "IOPT-TOOLS – A web based tool framework for embedded systems controller development using Petri nets"; FPL’2013 – International Conference on Field Programmable Logic and Applications; September 2-4, 2013, Porto, Portugal

[22] - Fernando Pereira, Filipe Moutinho, José Ribeiro, Luís Gomes; "Web Based IOPT Petri Net Editor with an Extensible Plug-in Architecture to Support Generic Net Operations"; IECON'2012 - 38th Annual Conference of the IEEE Industrial Electronics Society; October 25-28, 2012, Montreal, Canada[

[23] - Fernando Pereira, Filipe Moutinho, Luís Gomes; "Model-checking framework for Embedded Systems Controllers Development using IOPT Petri nets"; ISIE'2012 - 21th IEEE International Symposium of Industrial Electronics; May 28-31 2012, Hangzhou, China

[24] – Moutinho, Filipe; Gomes, Luís, “Asynchronous-Channels Whithin Petri Net-Based GALS Distributted Embedded Systems Modeling”, Industrial Informatics, IEEE Transactions on, vol 10, no.4, pp 2024,2033, Nov. 2014, DOI: 10.1109/TII.2014.2341933

[25] - D. A. Zaitsev. Compositional analysis of Petri nets. *Cybernetics and Systems Analysis*, 42(1):126{136, January 2006. DOI 10.1007/s10559-006-0044-0.

[26] - Tatsushi NISHI and Ryota MAENO. Petri Net Modeling and Decomposition Method for Solving Production Scheduling Problems. *Journal of Advanced Mechanical Design, Systems, and Manufacturing*, 1(2):262{271, 2007.

[27] - G. Bruno, R. Agarwal, A. Castella, and M. P. Pescarmona. CAB: an environment for developing concurrent applications. In *Applications and Theory of Petri Nets 1995, 16th*

*International Con-ference (ICATPN 1995)*, Jun. 1995.

[28] - Remigiusz Wi´sniewski, Andrei Karatkevich, Marian Adamski, Anikó Costa, Member, IEEE, and Luís Gomes, Senior Member IEEE; Prototyping of Concurrent Control Systems With Application of Petri Nets and Comparability Graphs, IEEE.

[29] - Olaf Kummer. A Petri Net View on Synchronous Channels. Petri Net Newsletter, 56:7–11, 1999.

[30] – World Wide Consortium (W3C),*http://www.w3.org/TR/REC-xml acedido pela última vez em: 19/09/2018*;

[31] – Libxml2, *http://xmlsoft.org/, acedido pela última vez em: 19/09/2018*;